Active and Passive Elec. Comp., 2002, Vol. 25, pp. 289–306

AN ADAPTIVE DIGITAL FUZZY ARCHITECTURE FOR APPLICATION-SPECIFIC INTEGRATED CIRCUITS

N. E. EVMORFOPOULOS and J. N. AVARITSIOTIS*

Department of Electrical and Computer Engineering, National Technical University of Athens, 9, Iroon Polytechniou St., Zographou, Athens, 15773, Greece

(Received 29 April 2002; In final form 3 July 2002)

A generic fuzzy logic system implementation in digital hardware is presented in this paper. The architecture developed is oriented to Application-Specific Integrated Circuits (ASICs), and its functionality is demonstrated through a typical ASIC for a simple control application. Furthermore, a distributed adaptation scheme is proposed for real-time environments. Performance/area tradeoffs for VLSI implementation are also discussed.

Keywords: Fuzzy logic; Digital integrated circuits; Control systems and applications

1 INTRODUCTION

The past few years have witnessed a rapid growth in the number and variety of applications of fuzzy logic that demonstrate a superior performance and cost-effectiveness to conventional approaches. The main principles underlying fuzzy logic have been extensively covered in the literature [1, 2]. Fuzzy logic originated [3, 4] as a tool for mathematical modeling of imprecision and uncertainty inherent in human reasoning. As such it can be used to introduce expert knowledge to a system in the form of rules and relationships specified in natural language. However, within the following years it has proved to be an extremely powerful way to map an input space to an output space with any degree of complexity and linearity. Recent results [5–9] show that a finite number of fuzzy rules can approximate any continuous I/O function with arbitrary accuracy.

Today applications of fuzzy logic cover such diverse fields as automatic control, consumer electronics, household appliances, data classification, decision analysis, expert systems, computer vision, signal processing, time-series prediction, information retrieval and so on. As potential applications grow, so does the need for efficient hardware implementations of fuzzy systems. Current fuzzy systems are built on general-purpose computing machines or programmable fuzzy processors designed to store and process fuzzy rules [10–12]. Although this approach offers flexibility and, sometimes, reduced development time, it often lacks the power to cope with real-time applications (*e.g.* image and speech processing) and is also associated with significant hardware overhead. On the other hand

* Corresponding author. Tel.: +3010-7722547; E-mail: abari@cs.ntua.gr

ISSN 0882-7516 print; ISSN 1563-5031 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/0882751021000042235

application-specific fuzzy hardware architectures [13, 14] aim at the highest implementation efficiency in terms of processing speed and silicon utilization.

Real-time applications would also benefit from (and in most cases mandate) control systems that exhibit adaptive nature by altering their behavior according to changes in operating and environmental conditions. On-chip adaptation is one of the most recent trends of research in the area of hardware systems in general and fuzzy hardware specifically. An attempt of including adaptive blocks at various stages of a fuzzy system is made within the framework of the proposed architecture.

Digital hardware has the advantage of being more accurate, robust and fault tolerant when compared to its analog counterpart. Furthermore, significant advances in the EDA-CAD field during the past few years have made the design, simulation, verification and implementation process of digital systems considerably more efficient and with shorter turnaround time than that of the analog systems. Finally, the majority of hardware available nowadays relies on digital operation and any new architecture would benefit significantly from compatibility to the vast array of digital systems, computers, peripherals and network protocols for data exchange purposes.

2 SYSTEM OVERVIEW

The proposed digital fuzzy architecture was incorporated into a controller ASIC for window-type air conditioners. It should be emphasized though that, with some trivial modifications, this architecture could suit any other fuzzy system implemented in digital hardware. The purpose of this particular application reported here is only to present the architectural mapping and provide insight on the important tradeoffs during the design process.

The ASIC receives one analog input from a thermistor sensor representing the ambient temperature and two digital inputs from two up/down buttons that define the temperature setting. Both temperature values have a resolution of 0.5 °C and a range of 16 °C to 32 °C. The ASIC also provides two pulse-width-modulated (PWM) outputs to two dc motors and one digital output to an on/off relay. The two motors are the Compressor which controls the temperature scale of the recycled air and the Fan which controls the fan speed. Each motor operates in one of the 4 possible modes (Auto-Cool-Heat-Off and Auto-High-Med-Low respectively) shown in Table I. Fuzzy control is established only for the Auto modes. The on/off relay is the Reverse Valve which controls the Cool/Heat operation of the Compressor. The ambient temperature analog value feeds an internal 8-bit ADC based on a successive approximation conversion technique which takes 16 clock cycles to finish. For a 512 Hz ADC clock the output data are valid every $512/16 = 32$ Hz and this frequency is used to sample a register that stores the converted value. The outputs to the dc motors are

TABLE I System Operation Summary.

Compressor		*Fan*	
Mode	*Operation*	*Mode*	*Operation*
Auto	Fuzzy	Auto	Fuzzy
Cool	$+30\%$, if $T > S$ 0%, if $T < S$	High	30%
Heat	-30%, if $T < S$ 0%, if $T > S$	Med	60%
Off	0%	Low	90%

formed by two internal 8-bit PWM timer modules which control two external transconductance power amplifiers that deliver the appropriate motor currents. The mean power applied to a motor load is defined by the pulse duty cycle which is taken as the percentage of the timer termination count to the timer maximum count (equal to $2^8 - 1 = 255$). Therefore, both outputs have a range of 0% to 100%. Their resolution is equal to an 8-bit partition of the $[0, 1]$ interval *i.e.* $1/256 = 0.0039$ or 0.39%. Finally, the Reverse Valve relay takes as input the sign bit of the Compressor pulse control word.

Other features of the ASIC, which will not be of any further concern here, include: (i) embedded IR decoder circuitry for remote control, (ii) a 12-hour timer for automatic termination or restarting, and (iii) optional connection to an external serial EEPROM for storage of the last system state (mode and temperature setting).

3 FUZZY LOGIC ALGORITHM

The primary concept underlying fuzzy logic and which distinguishes it from conventional approaches, is that of fuzzy sets. These are sets which can contain elements with only partial degree of membership. A formal definition states that a fuzzy set A is the set of ordered pairs $A = \{x, \mu(x) \mid x \in X\}$, where $\mu(x)$ is the membership function (MF) of x in A and X is the universe of discourse where A is defined. The MF maps each element $x \in X$ to a membership value between 0 and 1, specifying the degree to which x belongs to the fuzzy set A in X.

Most often, MFs have the form of a radial-basis function (RBF) which has a maximum of 1 when the distance between the input and the center of the fuzzy set is 0 and decreases as this distance increases. Out of the many MF types available [15], one of the best choices is the Gaussian RBF which is defined as $\mu(x) = \exp(-(x - c)^2/2\sigma^2)$, where c and σ are the center and width (or spread) of the fuzzy set A respectively. Gaussian MFs have some very desirable properties such as local spatial and frequency content and smooth and infinitely differentiable output. We will also use a 2-sided variation of the Gaussian MF, which is characterized by different spread constants σ_1 and σ_2 for each side of the center.

A fuzzy inference system (FIS) is a system which makes use of fuzzy logic in order to perform an I/O mapping. Figure 1 depicts a FIS and its main constituent subsystems (fuzzifier, rule base and inference engine). In order to design a FIS we have to define the I/O variables and associated linguistic terms and then specify the rules that govern the system operation.

The input variables taken into account by the proposed FIS are the ambient temperature T, the absolute difference $|T - S|$ between the ambient temperature and the temperature

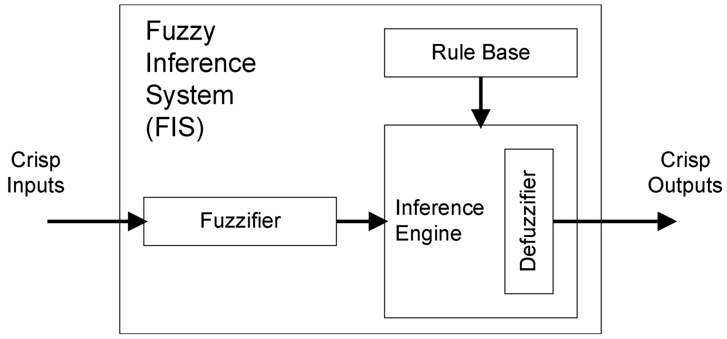

FIGURE 1 Fuzzy inference system (FIS) block diagram.

setting S, and their variations with time *i.e.* $\mathrm{d}T/\mathrm{d}t$ and $\mathrm{d}(|T-S|)/\mathrm{d}t$. The variations indicate the speed at which the variables T and $|T-S|$ are changing. Introducing them as inputs can give the system more accuracy and stability and provides an additional means of maintaining the ambient temperature near the setting one when they are already close enough. Variations are calculated at each new incoming temperature, as the difference between the current value and the previous value stored in an appropriate register. They have a range of $-4\,^\circ\mathrm{C}$ to $4\,^\circ\mathrm{C}$ and a resolution of $0.5\,^\circ\mathrm{C}$. All input values must be normalized in order to meet a full-range 8-bit partition of the universe of discourse. The above actions are illustrated in Figure 2.

Each input and output variable is characterized by a number of linguistic terms that cover the variable's entire range of values (universe of discourse). Each linguistic term is represented by a fuzzy set with a particular MF. It is always necessary to define at least two fuzzy sets on each variable because the definition of a linguistic term is always relative to the neighboring ones.

The choice of linguistic terms and corresponding Gaussian-type MFs associated with the input variables T, $|T-S|$ and their variations is shown in Figure 3 (only one variation variable is shown as both have the same partition of input space). MFs should have enough overlap with each other in order to provide a smooth output transition between adjacent regions. Sigmoid functions are also employed as MFs because they are more appropriate for representing concepts that lie at the extremes of the universe of discourse. They consist of a Gaussian part at one side and a constant part of 1 at the other side of their central value. Because the definition of linguistic terms that are far from zero is often fuzzier than the neighboring ones, it is a good practice to distribute the $|T-S|$, $\mathrm{d}T/\mathrm{d}t$ and $\mathrm{d}(|T-S|)/\mathrm{d}t$ MFs so that there is a fine resolution near zero and a progressively coarser resolution as we get further away. Moreover, in order to get good input space coverage, the spread constants must be at least equal to the input resolution ($0.5\,^\circ\mathrm{C}$ in the current case). However, they should be significantly smaller than the distance across the whole input space ($16\,^\circ\mathrm{C}$ for the temperature inputs or $8\,^\circ\mathrm{C}$ for the variations) so as not to lose accuracy.

The fuzzifier block determines the degree to which input variables belong to each of the appropriate linguistic terms via the corresponding fuzzy set MFs. It generally maps crisp numerical values limited within the input variable's universe of discourse to degrees of membership within the interval $[0, 1]$, which is divided into 256 values for 8-bit mapping.

Input/Output variables and linguistic terms are associated with rules which specify a fuzzy relation R: $X \rightarrow Y$ in the I/O space. Rules may be provided by experts or can be extracted from numerical data. They are expressed as a collection of "IF $\langle$input$\rangle$ THEN $\langle$output$\rangle$" statements. Each rule is divided into two parts, the premise (or antecedent) and the consequent.

The rule premise applies logical operators such as AND and OR to the membership values extracted from the fuzzifier. These operators are not the conventional Boolean ones but their fuzzy extensions representing fuzzy intersection and fuzzy union respectively. In general, any T-norm and T-conorm operators can be used [16] but the min and max operators, defined by $\mu_{A\cap B} = \min(\mu_A(x), \mu_B(x))$ and $\mu_{A\cup B} = \max(\mu_A(x), \mu_B(x))$, are the most suitable for digital implementation. As it will become clear later, in order to have a more effective mapping of the rule premise onto digital hardware, it is desirable to have only AND operators. This can be accomplished by the following generalization of De Morgan's law [17]: $\max(\mu_A(x), \mu_B(x)) = 1 - \min(1-\mu_A(x), 1-\mu_B(x))$. The evaluation of the rule premise determines the rule firing strength f. This is multiplied by a rule confidence v that acts as a weight between the premise and the consequent, indicating the rule's contribution to the output. Apart from their utilization as weights, rule confidences play a vital role in the formation of a dynamic rule structure. Specifically, we initially map all possible rules to

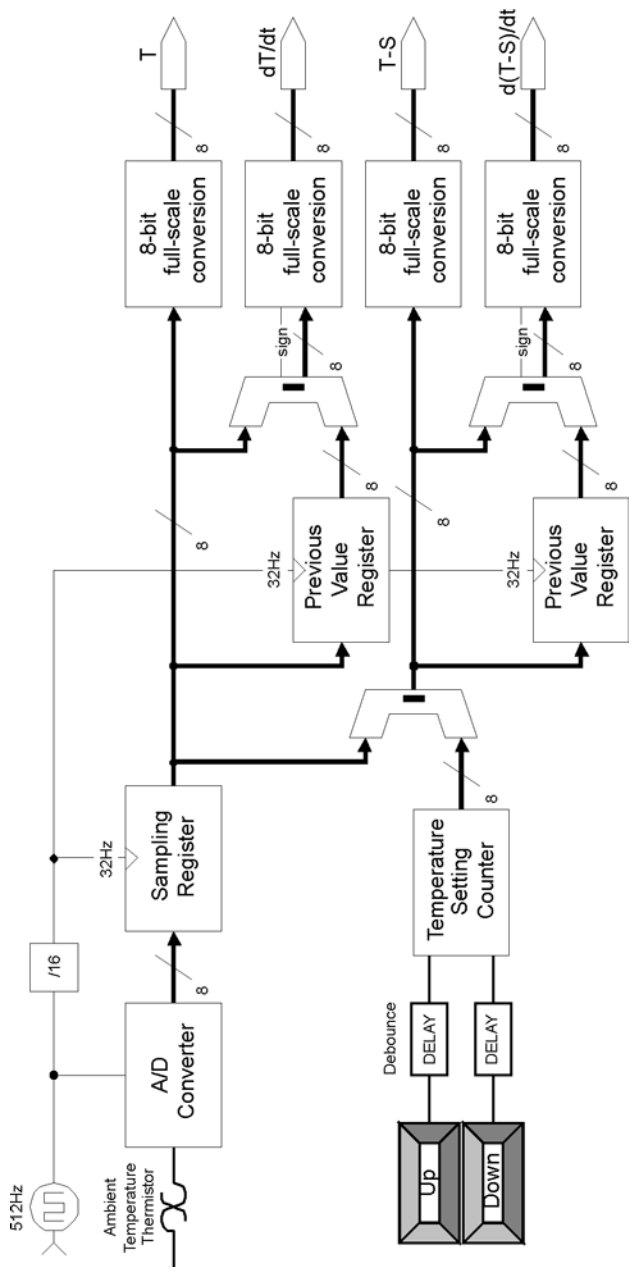

FIGURE 2 Formation of inputs to the fuzzy inference system.

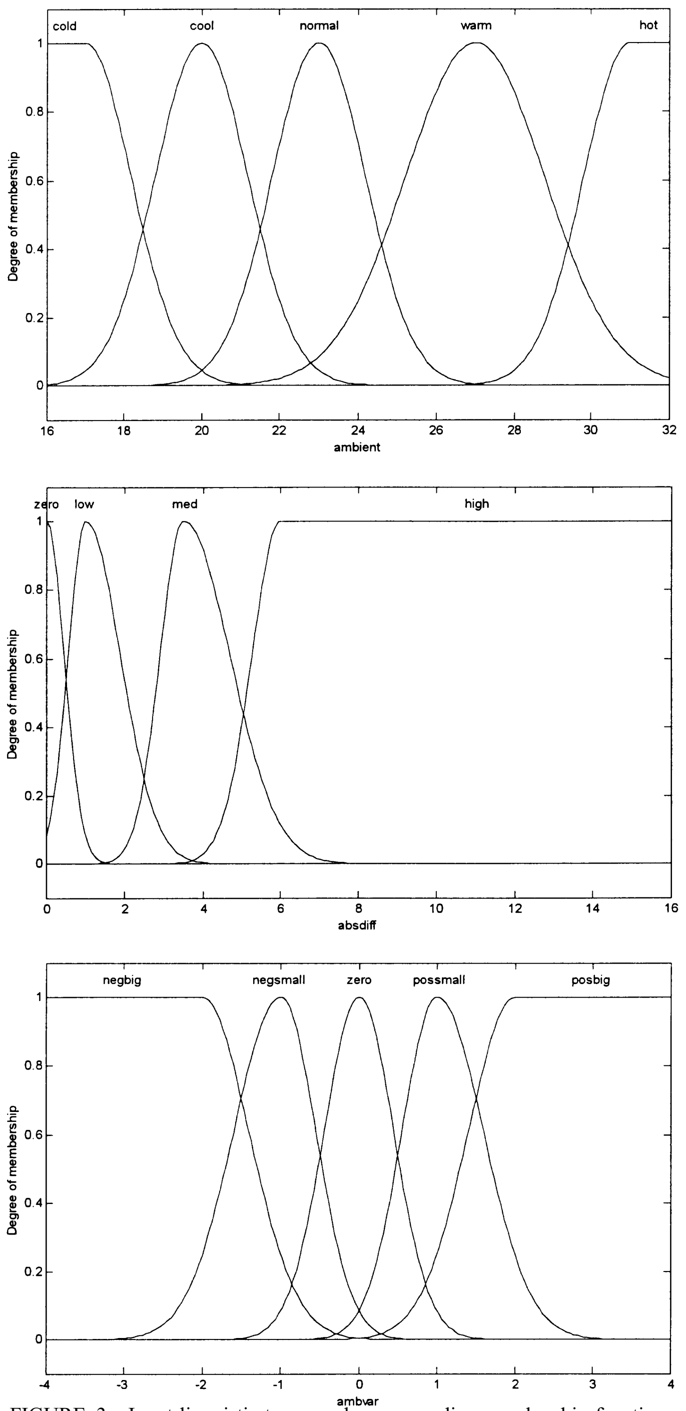

FIGURE 3 Input linguistic terms and corresponding membership functions.

hardware and let the values of the rule confidences at the current adaptation loop determine the inactive and active rules (along with their degree of activity).

The rule base encoding "expert" knowledge for temperature control in the proposed FIS is shown in Table II. Rules for the Compressor motor take into account the variables T and $\mathrm{d}T/\mathrm{d}t$ whereas rules for the Fan motor make use of the $|T - S|$ and $\mathrm{d}(|T - S|)/\mathrm{d}t$ variables. It should be noted that if both the Compressor and Fan motors are in Auto mode, the temperature setting S is made equal to the "normal" linguistic term central value (initially set to 23 °C) and cannot be altered manually. Each rule corresponds to a fuzzy patch in I/O space which reflects the rule's fuzziness (or uncertainty). The complete rule base forms a set of overlapping patches, which define a 3D surface in I/O space (Fig. 4).

The inference engine is the heart of the FIS algorithm. It connects input and output linguistic terms as defined by the rule base and handles the way in which the numerical results of individual rules are combined to yield a crisp output.

Two major fuzzy inference models [18–20] are most commonly used in practice. These are the Mamdani and Sugeno models, which only vary in the way outputs are determined. The Mamdani model expects the output linguistic terms in the rule consequent to be fuzzy sets. The corresponding MFs are clipped (or scaled) at the rule premise firing degree and then combined through an OR (or sum) operator. The resulting fuzzy set is passed through a defuzzifier block, usually performing a centroid computation, in order to yield a crisp value that best represents the fuzzy set. The Sugeno model, which is adopted in the current FIS, employs crisp values z_i (called fuzzy singletons) rather than distributed fuzzy sets to represent the output linguistic terms. The output z is a weighted average of z_i over all output terms i, *i.e.* $z = \sum_i w_i z_i / \sum_i w_i$, where w_i is equal to $f_i v_i$ for the rules with term i in their consequent part.

It must be pointed out that if more than one rules have the same output linguistic term, they must be aggregated through an OR operator in order to obtain the final w_i.

The choice of output linguistic terms and corresponding singletons is shown in Figure 5. In most cases, singleton (instead of fuzzy) output values are completely sufficient for a given problem's needs. It can be shown [15] that the output of a Sugeno model is a smooth function of its input variables as long as the neighboring MFs in the premise have enough overlap. This means that the shape (fuzzy or singleton) of the MFs in the consequent does not have a decisive effect on the smoothness of the I/O surface. Moreover, under the constraint

TABLE II Rule Bases for Compressor and Fan Outputs.

	Negbig	*Negsmall*	*Zero*	*Possmall*	*Posbig*
Compressor-Auto					
T			$\mathrm{d}T/\mathrm{d}t$		
Cold	– Big	– Big	– Med	– Small	– Tiny
Cool	– Big	– Med	– Small	– Tiny	– Zero
Normal	– Tiny	Zero	Zero	Zero	+ Tiny
Warm	Zero	+ Tiny	+ Small	+ Med	+ Big
Hot	+ Tiny	+ Small	+ Med	+ Big	+ Big
Fan-Auto					
$\|T - S\|$			$\mathrm{d}(\|T - S\|)/\mathrm{d}t$		
Zero	Zero	Zero	Zero	Tiny	Small
Low	Zero	Zero	Tiny	Small	Med
Med	Zero	Tiny	Small	Med	Big
High	Tiny	Small	Med	Big	Big

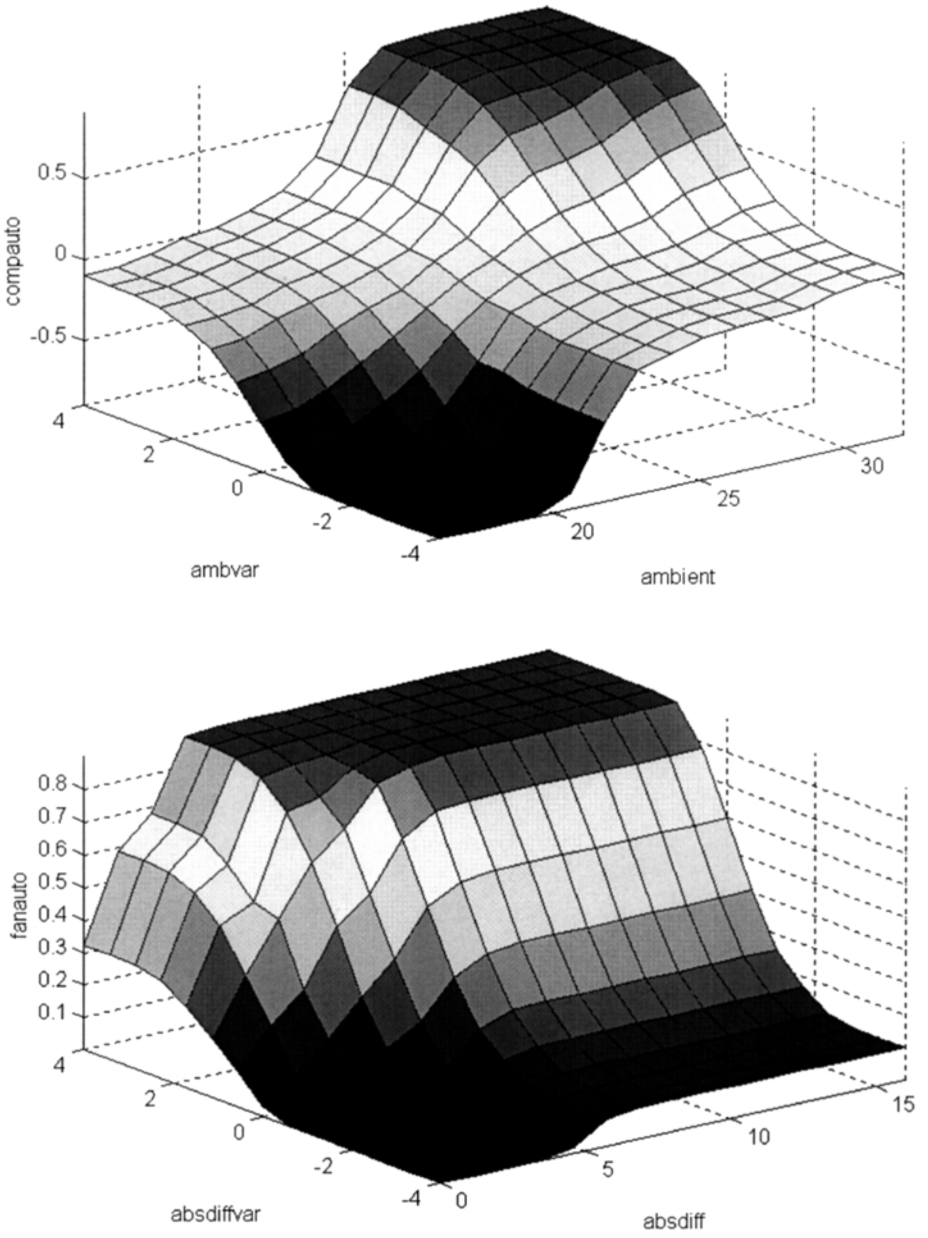

FIGURE 4 I/O surfaces for Compressor and Fan outputs.

of symmetric output MFs with equal spread constants, a Mamdani model is numerically equivalent (with a 2nd decimal point degree of accuracy) to a zero-order Sugeno model, with each output MF modeled as a singleton situated at its center. The Sugeno FIS model obviates the need for a defuzzifier block and thus greatly simplifies the hardware and computational overhead, as we take the weighted average of a few data points rather than integrating across a 2D function.

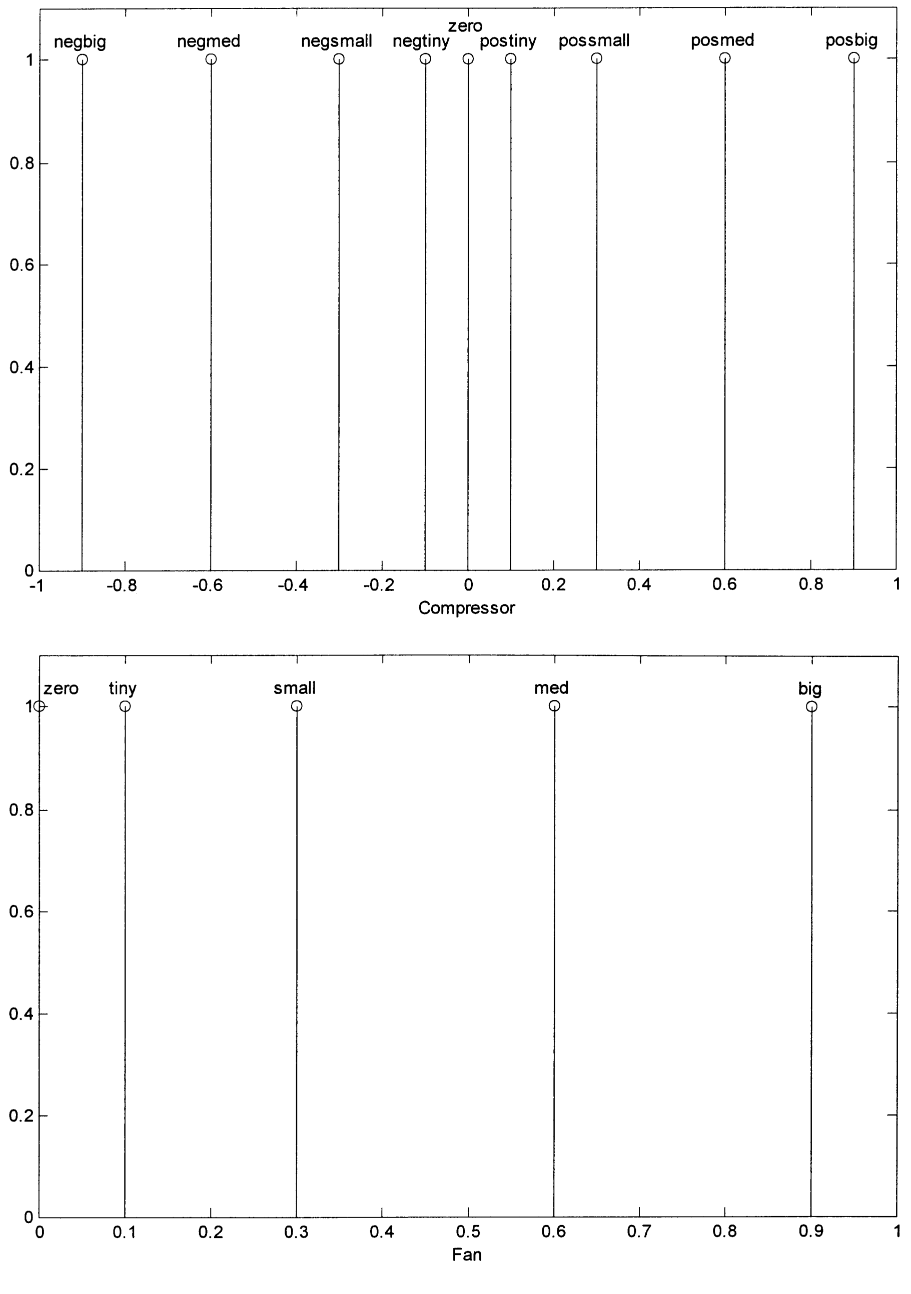

FIGURE 5 Output linguistic terms and corresponding singleton values.

4 ADAPTATION ALGORITHMS

The algorithms employed to realize the adaptive fuzzy system are taken from the neural network realm. Two major disciplines are usually followed: supervised learning and self-organization. In our case, it is not feasible to carry a supervised approach because of its off-line learning pattern and the unavailability of a pre-arranged set of representative training data. On the other hand, self-organized learning is simpler and well suited to on-line operation as the parameters are only adjusted once each cycle.

Self-organizing techniques work by clustering data into groups called classes, the members of which resemble one another. Each I/O term or rule is associated with a quantization vector of a particular class. During every cycle, new data stimulate and adjust the quantization vectors of the classes that match the data most closely (active or "winning" vectors). The statistical properties of data over time dynamically form the vector clustering. This approach serves to allocate the FIS resources (I/O MF parameters and rule structure) efficiently by covering only those regions of the I/O space where data are present.

The MF parameters allowed to adapt are the centers c_i and z_i, which are adjusted with a variation of Kohonen's feature maps algorithm [21] *i.e.* $\Delta c_i = a_1 y_x(x - c_i)$ and $\Delta z_i = a_2 y_z(z - z_i)$, where x, z are the input and output respectively, y_x, y_z are their transformed values through a radial-basis MF with centers c_i, z_i, and a_1, a_2 are the learning rate parameters with positive values near zero. Each quantization vector moves towards the input data to an extent depending on its distance from them as defined by the radial-basis MF. The distinction from the original algorithm is that we do not search for a winner-take-all (WTA) vector, nor define a neighborhood to train around WTA. The quantization vectors that correspond to the input MF centers c_i are adapted during the normal feed-forward stage when the input data pass through the MF block. On the other hand, in order to adapt the vectors associated with the output singletons z_i, the outputs should pass through a radial-basis MF in a feedback stage after the formation of their final crisp value. Each z_i is passed through the same RBF which is chosen to be of Gaussian-type with a spread constant equal to $\sigma = 0.9$.

The rule structure (active and inactive rules) is defined by the rule confidences v_i. The values of v_i are adjusted by the reinforcement algorithm $\Delta v_i = a_3 f_i - d v_i$, where a_3 is a learning rate and d is a decay rate of v_i, both existing within the interval $[0, 1]$. During every cycle, the rules which fire most strongly (indicated by their firing strength f_i) are reinforced while the others are left to decay. In other words, rules are made to fire in a greater or lesser extent (defined by an increase or decrease in v_i) according to their current firing strength.

5 DIGITAL HARDWARE ARCHITECTURE

We adopted a 3-layer architectural mapping of the FIS in Table I, with each layer corresponding to a FIS subsystem. The layers operate in sequence, receiving inputs from the previous layer and providing outputs to the next one. They also realize the distributed adaptation scheme presented in Section 4.

Within each subsystem, all operations are controlled by a Finite State Machine (FSM) which consists of a PLA and a state register. The FSM generates the appropriate timing signals that activate the various parts of the subsystem whenever appropriate. The FSM control sequences of all 3 FIS subsystems are shown in Figure 6. It should be noted that, in order to perform the conditional redirections, the FSMs require random-access information (stored in a register file) about the number of I/O variables and the number of corresponing MFs, as well as the number of input MFs associated with a particular output variable via the rule base.

The fuzzifier subsystem architecture is depicted in Figure 7. Two counters run through all input variables and their partition in membership functions. The input variable counter

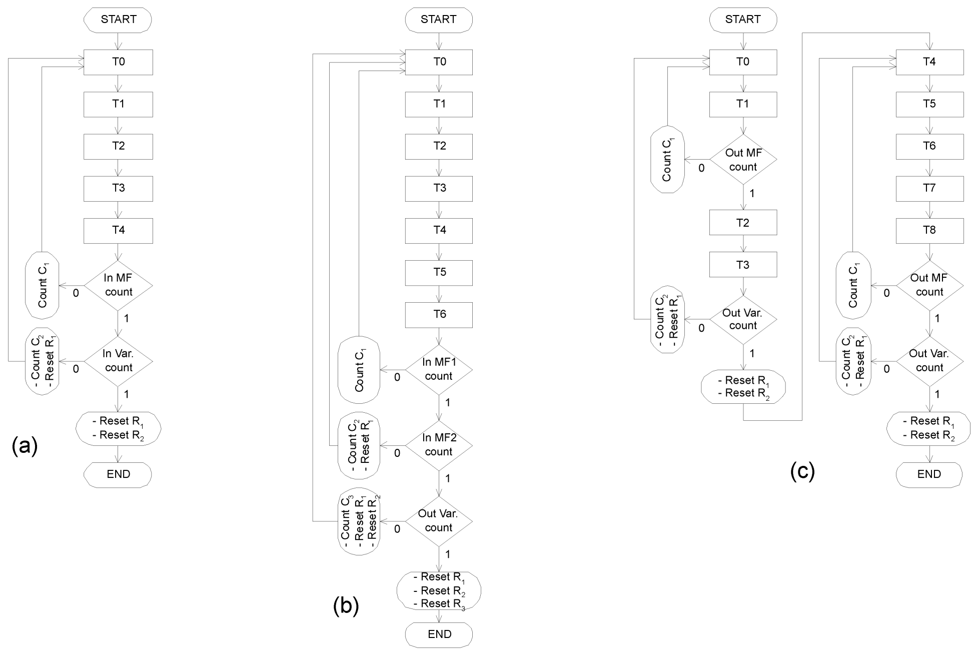

FIGURE 6 Finite State Machine (FSM) control of (a) fuzzifier, (b) rule base and (c) inference engine.

drives the selection lines of a 32×8 multiplexer which fetches the corresponding input. All MF centers and spread constants are stored in two 19×8-bit register files with shift-down capability. The MF computation is most efficiently realized by a lookup table (LUT) approach [13], which also has the advantage of being compatible with various MF types, thus adding some flexibility to the FIS implementation. Specifically, we prestore the $\mu(x) = \exp(-x^2/2\sigma_i^2)$ values for various spread constants σ_i in a ROM or RAM-type LUT and obtain the degree of membership by sequentially feeding the $x - c_i$ values for all MFs to the address lines. The 3 most significant address bits select one spread constant out of a total number of 6. The 000 combination is reserved for the extreme sigmoid functions where all memory locations should return a constant of 1. A 6×3 multiplexer aids in the realization of 2-sided MFs by selecting the left or right σ_i value from the register file, according to the sign of the $x - c_i$ operation. In order to reduce the memory size requirements we only store the non-zero positive values, as they are symmetric and cover just a few points around the zero-valued center. The least significant bits of the $x - c_i$ operation enter the memory address lines, while the most significant bits decide if the result is zero or non-zero. This arrangement is illustrated in Figure 8. The degree of membership as returned by the LUT is shifted down in another register file that stores the MF outputs and which, at the end of the fuzzification stage, provides the inputs to the next subsystem (rule base) of the FIS.

The adaptation block of the fuzzifier calculates the new MF centers at each cycle with the algorithm cited in Section 4. The necessary numerical operations are performed with a 16×8-sized ROM-type multiplier LUT which holds the outcome of multiplication for all binary combinations in an 8-bit partition of the $[0, 1]$ interval. The computation of the algorithm is executed in two stages of memory accessing, with the partial and final result stored in an 8-bit register. The MF center adjustment value Δc_i is added to the current c_i

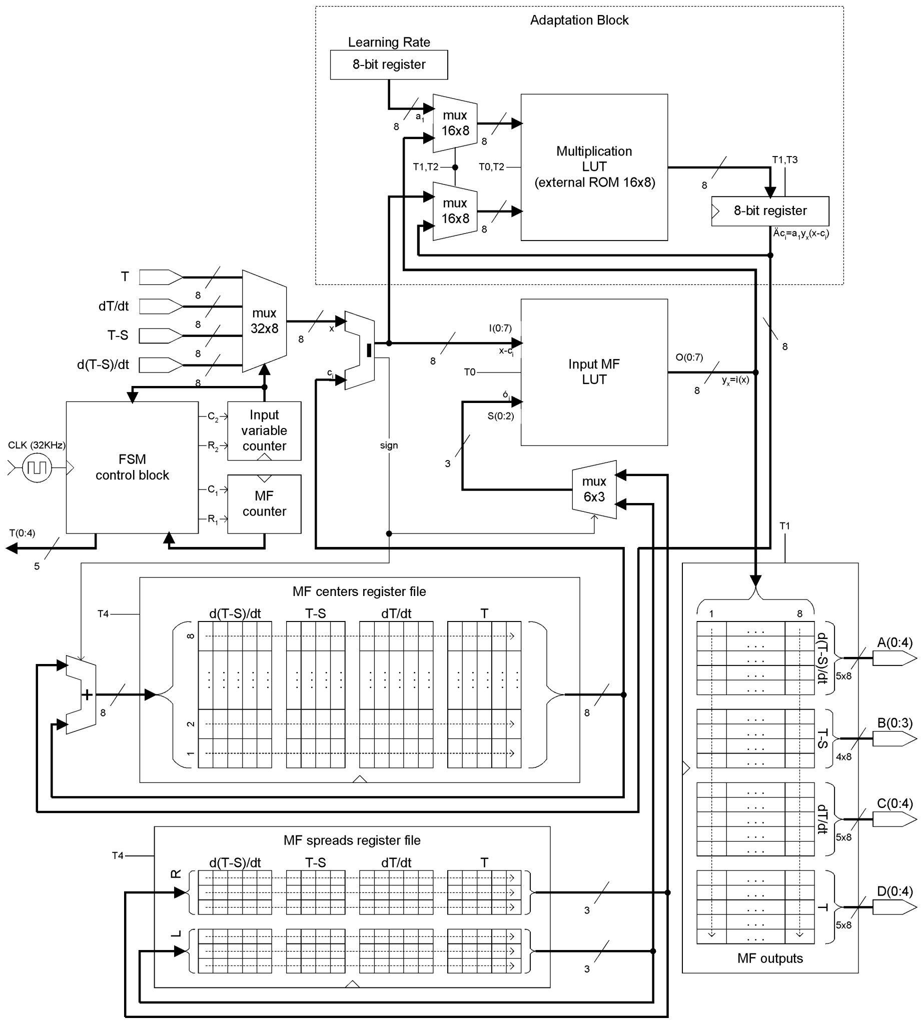

FIGURE 7 Fuzzifier block architecture.

value, which is fed back while the next c_i is shifted down. It should be pointed out that while a_1 and $y_x = \mu(x)$ lie within the interval $[0, 1]$ and $(x - c_i)$ lies within the universe of discourse of the input (denoted by $[0, \mathrm{U}]$), the calculation of Δc_i will correctly return a result within $[0, \mathrm{U}]$. Indeed, provided that a full-range 8-bit partition was followed in both $[0, 1]$ and $[0, \mathrm{U}]$, the above calculation via the multiplication LUT will correspond to $a_1 y_x(x - c_i)/U = \Delta c_i/U$ in $[0, 1]$, which is equal to the correct Δc_i when the latter is considered in $[0, \mathrm{U}]$.

The rule base implementation is shown in Figure 9. Two counters (which may well be the ones encountered in the previous subsystem that is now inactive) run through all the MF combinations of the two input variables that form the rule premise while another one is assigned to count the output variables of the consequent. The aim is to provide a full rule base connectivity. An arrangement of multiplexers controlled by the counters is used to select two of the outputs from the previous stage and pass them through a min block (Fig. 10a [13]) which realizes the fuzzy AND operation. The rule firing strength f_i thus

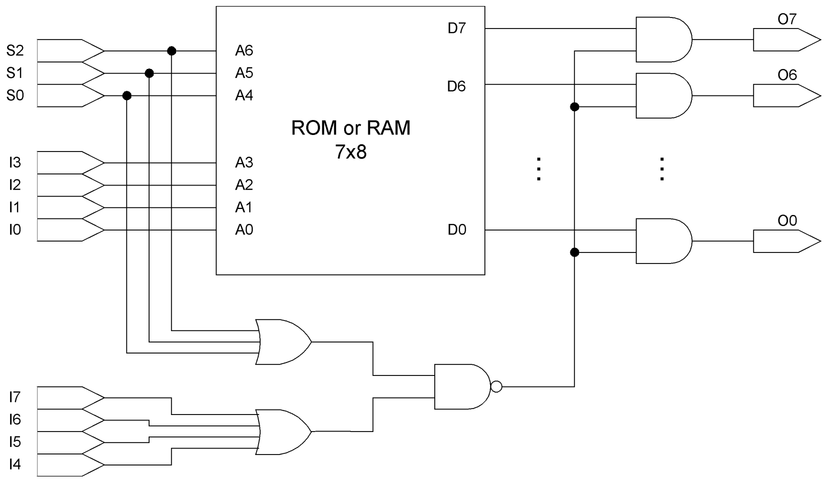

FIGURE 8 Detail of the input MF block.

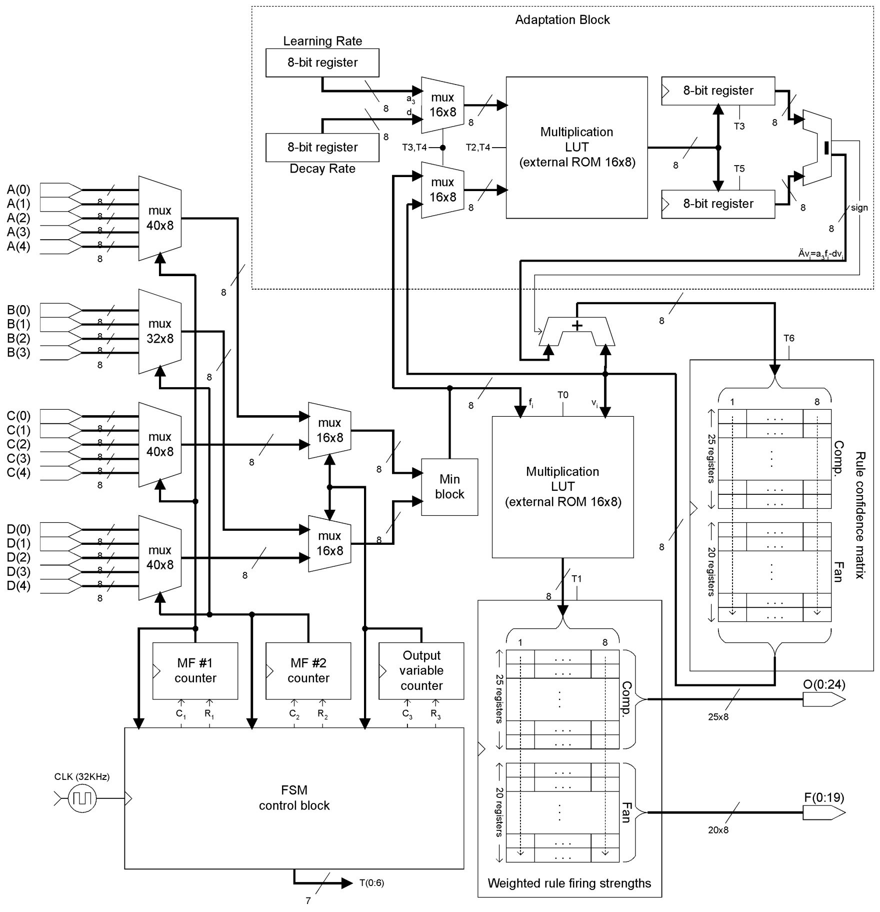

FIGURE 9 Rule base architecture.

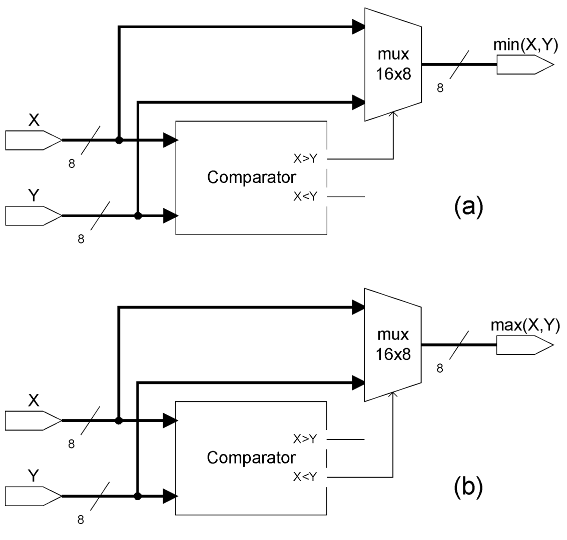

FIGURE 10 Implementation of the (a) min and (b) max blocks.

formed is multiplied via the multiplication LUT with the corresponding rule confidence v_i which is stored in a 45×8-bit shift-down register file. The weighted firing strengths as returned by the multiplication LUT are shifted down in another register file, providing the inputs to the inference engine subsystem.

The adaptation block of the rule base computes the rule confidence adjustment values Δv_i at each cycle using the corresponding algorithm in Section 4. It makes use of the multiplier LUT to calculate the two partial results $a_3 f_i$ and dv_i, which are then subtracted to provide Δv_i.

The inference engine architecture is shown in Figure 11. Two counters run through all output variables and their partition in singleton values. A rule-to-output connectivity block receives the w_i values of all rule premises from the previous subsystem and redirects them to match the corresponding output singletons z_i in the order they are stored in a 14×8 register file. The aforementioned block contains the appropriate max blocks (Fig. 10b) that implement the fuzzy OR operation whenever more than one rules have the same consequent. A substantial step towards a flexible rule structure would be to replace the rule-to-output connectivity block with a reconfigurable array of min blocks with multiplexed inputs selected by control words. The $\sum_i w_i z_i$ and $\sum_i w_i$ values are sequentially formed in two 12-bit registers which feed back the stored partial results to two adders for the next addition in the chain. At the end, the crisp output value z is generated by a 24×9-sized ROM-type division LUT which holds the outcome of division for all binary combinations in an 8-bit partition of the output universe of discourse ($[0, 1]$ in this case). The LUT and registers have been designed to handle a larger word size than the standard 8-bits, due to potential overflow that may arise in the preceding sequential addition. Finally, the sign from the Compressor output is used as the control bit for the Reverse Valve relay.

After the crisp value generation, the outputs are fed to the adaptation block which is identical to the one used in the fuzzifier subsystem, in order to calculate the adjustment in the singleton values Δz_i.

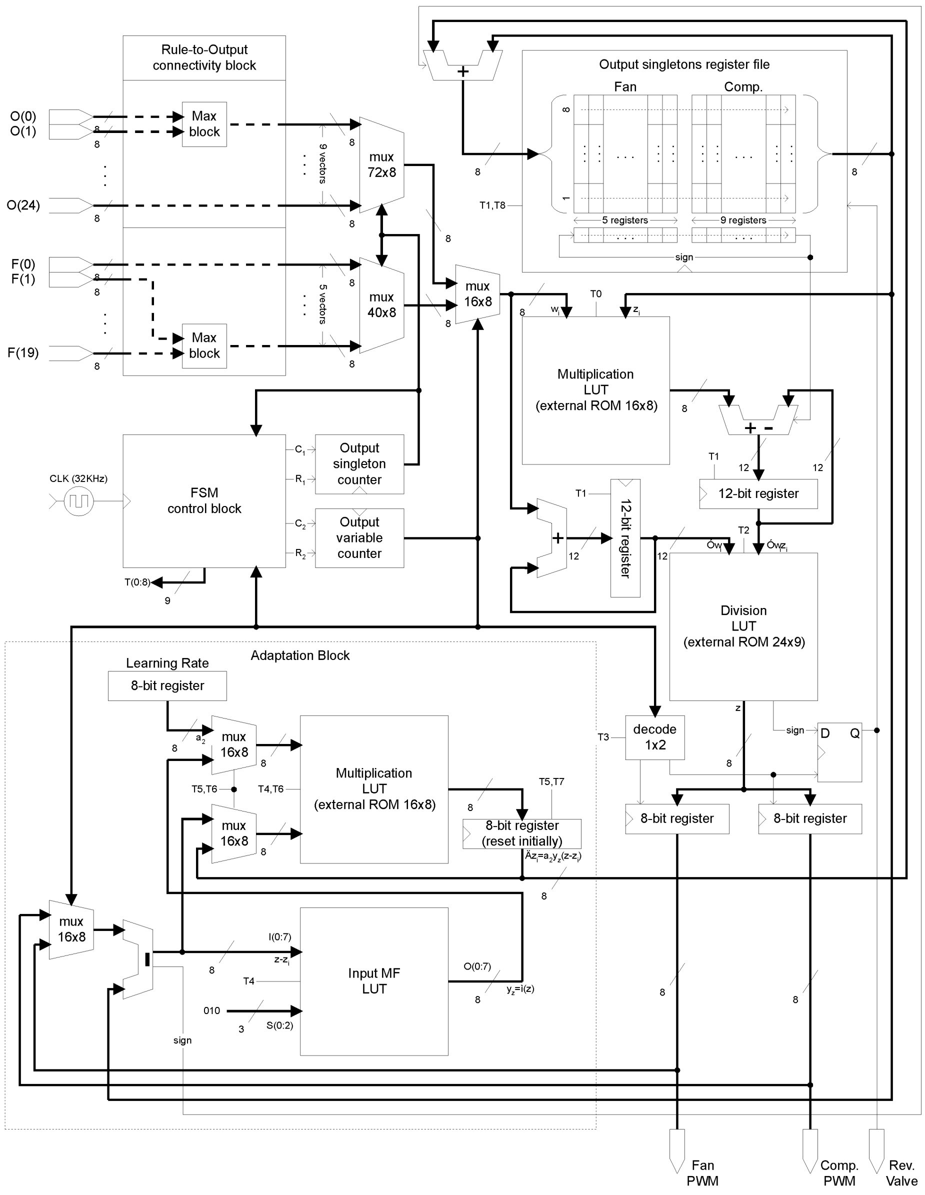

FIGURE 11 Inference engine architecture.

6 VLSI IMPLEMENTATION AND PERFORMANCE ISSUES

The ASIC was fabricated in 0.7 μm double-metal single-poly *n*-well CMOS technology. It incorporated approximately 35,000 gates and had 44 I/O pins. A layout and a photo are shown in Figure 12. The master clock had a frequency of a 32 KHz and was chosen so that all three FIS operations are completed before the next 32 Hz sample of the input. It was not possible to embed all three memory-type LUTs within the ASIC. Since the largest memory cell available for internal use was 12×8, only the input MF ROM was built

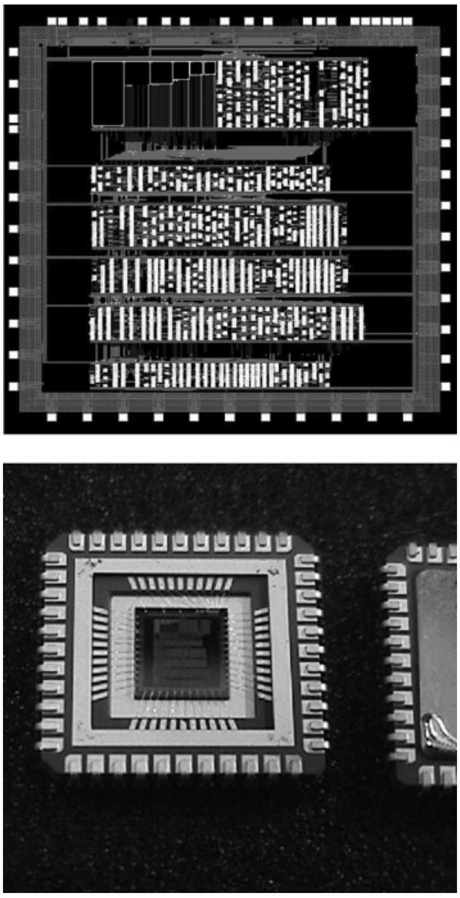
FIGURE 12 (a) Layout and (b) photo of the ASIC.

internally while both multiplication and division ROMs were placed outside the ASIC. A multiplexed bus instead of a tristate one was employed to access each memory, whenever required at the various FIS stages. The choice is justified due to the lower power consumption (a multiplexer does not have to drive the output load capacitances of the other drivers as "seen" in a tristate bus) as well as the avoidance of bus contention which is always a concern in tristate buses. The data stored in the memory locations were easily generated through a computer program running in a loop so as to create the appropriate file to be exported to the memory cell compiler (internal case) or the ROM programmer (external case).

In scientific and technical literature two performance measures are often used to characterize a fuzzy digital implementation. These are the number of fuzzy inferences per second (FLIPS) and the number of elementary fuzzy operations per second (min or max). However, the above parameters do not always provide an objective measure of the real implementation efficiency and throughput as they are technology-dependent. Therefore we propose to characterize the current FIS architecture by means of the total number of clock cycles from the moment of providing the input variables until the generation of the crisp value at the output.

The fuzzification, rule base and inference engine stages last $5 \cdot \sum_{i=1}^{n} M_i$, $7 \cdot \sum_{j=1}^{m} \times (\prod_{i=1}^{n_j} M_i)$ and $2 \cdot m + (2 + 5) \cdot \sum_{j=1}^{m} S_j$ clock cycles respectively, where n, m: total number of input and output variables respectively, M_i, S_j: number of MFs or singletons for input variable i or output variable j respectively, n_j: number of input variables associated with output variable j. The above figures yield a total number of cycles equal to $5 \cdot [\sum_{i=1}^{n} M_i] + 7 \cdot [\sum_{j=1}^{m}(\prod_{i=1}^{n_j} M_i)] + 7 \cdot [\sum_{j=1}^{m} S_j] + 2 \cdot m$.

It is interesting to notice that the adaptation process takes up $2 \cdot [\sum_{i=1}^{n} M_i] + 4 \cdot [\sum_{j=1}^{m}(\prod_{i=1}^{n_j} M_i)] + 5 \cdot [\sum_{j=1}^{m} S_j]$ clock cycles, which comprise 40% of the fuzzification (states T2–T3), 57% of the rule base (states T2–T5) and approximately 70% of the inference engine (states T4–T8) FSMs. It is obvious that, while the adaptation discipline imposes very small hardware requirements, it constitutes a substantial timing overhead (over 50% of the total). In addition, there are extra time savings involved if we can part with the adaptation feature. Specifically, the two states following the memory access T0 in the two first blocks can be merged (or pipelined) into one state *i.e.* while the memory result is being processed, the appropriate subsequent value can be shifted down in the register file. This leads to a total of $2 \cdot [\sum_{i=1}^{n} M_i] + 2 \cdot [\sum_{j=1}^{m}(\prod_{i=1}^{n_j} M_i)] + 2 \cdot [\sum_{j=1}^{m} S_j] + 2 \cdot m$ clock cycles.

Substantial performance improvement with little extra hardware involved can be achieved if we take advantage of the data dependencies in the input universe of discourse. More specifically, the degree of overlap of the input MFs is usually low which means that most of the fuzzy rules are not activated. Therefore, the basic idea is to route through the rule base subsystem only those rules whose firing strength f_i is non-zero, since the other rules have no effect on the output whatsoever. In order to realize this, special rule selection circuitry must be used, which mainly comprises of a flag register file that stores the indices of the non-zero input MFs (along with their total number) while their values are shifted down in the output register file of the fuzzifier. In the rule base subsystem, the indices stored in the flag register file drive the selection lines of the multiplexer arrangement, while the two MF counters now run through them in order to create all combinations of non-zero MFs. The multiplication LUT result is not shifted down to the firing strengths register file anymore, but must be written at the position currently indicated by the flag register. Therefore, random access to the output register file should also be provided. The timing figures of the fuzzifier and inference engine subsystems remain unchanged, but the rule base subsystem may exhibit significant reduction in the total number of clock cycles. The difference in speed becomes even more pronounced as the number of rules increases. Unfortunately, it cannot be quantified as it heavily depends on the MF degree of overlap.

The proposed architecture was designed with the minimum hardware requirements in mind. If even more throughput is required at the expense of a significant increase in hardware and interconnection complexity, one can turn the operation of the first two subsystems into parallel. Specifically, a number of input MF blocks equal to the total number of MFs and a number of min blocks equal to the total number of possible rules must be provided in the fuzzifier and rule base subsystems respectively. In the latter, the rule-weighting feature should also be discarded. These two subsystems would then produce a result in just 1 clock cycle each. However, the generation of a crisp output in the inference engine is the only part of the FIS that cannot be mapped in parallel hardware (unless we have a large number of multiplication LUTs which is impractical), as it involves numerical operations such as multiplication, addition and division on sequential binary vectors. Therefore it is considered to be a bottleneck that degrades the overall system potential, even if a Sugeno FIS is employed which avoids the integral-type defuzzification.

A final suggestion for performance improvement would be simply to increase the master clock frequency. Caution, however, has to be taken in multi-input max blocks which are commonplace within the rule-to-output connectivity block. The implementation of these blocks breaks up into a cascade of two-input max blocks which can result in significant delays and operation failures when high-frequency clock is involved. A good practice in such a situation is to generate carry look-back information within the multi-input max block [22] in much the same way binary full-adders incorporate carry look-ahead generation.

7 CONCLUSION

The superior performance that fuzzy logic usually exhibits can be attributed to the smoothness of the I/O surface, particularly in transition areas where input MFs overlap. This was proved in practice by the remarkably smooth transitions of motor speed outputs as the inputs varied. Clearly, as can be verified by closer examination of Figure 4, a highly complex nonlinear I/O relationship is required and the only way to readily obtain a model-free estimate of this is fuzzy logic.

Fuzzy logic implementations on software-based general-purpose computers are not able to reach the timing figures that dedicated hardware exhibits. One major cause is the CPU execution style, which sequentially fetches instructions and data from the program memory before proceeding to the main algorithm computation. Instead, dedicated hardware processes directly the data with the fuzzy functional blocks that have been described.

The architecture proposed in this paper is a dedicated hardware design which may suit any FIS with virtually no changes in the main FSM blocks and LUTs, and only minor alterations limited to the rule connectivity block structure and the sizes of counters, multiplexers and register files.

References

[1] Zadeh, L. A. (1988). Fuzzy logic. *IEEE Computer*, **1**(4), 83–93.
[2] Mendel, J. M. (1995). Fuzzy logic systems for engineering: a tutorial. *Proc. IEEE*, **83**(3), 345–377.
[3] Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, **8**(3), 338–353.
[4] Zadeh, L. A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. Systems, Man and Cybernetics*, **3**(1), 28–44.
[5] Kosko, B. (1991). *Neural Networks and Fuzzy Systems*, Prentice-Hall.
[6] Zeng, X. J. and Singh, M. G. (1994). Approximation theory of fuzzy systems: SISO case. *IEEE Trans. Fuzzy Systems*, **2**(2), 162–176.
[7] Zeng, X. J. and Singh, M. G. (1995). Approximation theory of fuzzy systems: MIMO case. *IEEE Trans. Fuzzy Systems*, **3**(2), 219–235.
[8] Landajo, M., Rio, M. J. and Perez, R. (2001). A note on smooth approximation capabilities of fuzzy systems. *IEEE Trans. Fuzzy Systems*, **9**(2), 229–237.
[9] Mitaim, S. and Kosko, B. (2001). The shape of fuzzy sets in adaptive function approximation. *Trans. Fuzzy Systems*, **9**(4), 637–656.
[10] Kandel, A. (1997). *Fuzzy Hardware: Architectures and Applications*, Kluwer.
[11] Gabrielli, A. and Gandolfi, E. (1999). A fast digital fuzzy processor. *IEEE Micro*, **22**(1), 68–79.
[12] Ascia, G., Catania, V. and Russo, M. (1999). VLSI hardware architecture for complex fuzzy systems. *IEEE Trans. Fuzzy Systems*, **7**(5), 553–570.
[13] Hung, D. (1995). Dedicated digital fuzzy hardware. *IEEE Micro*, **15**(3), 31–39.
[14] Jou, J. M., Chen, P. D. and Yang, S. F. (2000). An adaptive fuzzy logic controller: its VLSI architecture and applications. **8**(1), 52–60.
[15] Jang, J. S. R. and Sun, C. T. (1995). Neuro-fuzzy modeling and control. *Proc. IEEE*, **83**(3), 378–406.
[16] Dubois, D. and Prade, H. (1997). *Fuzzy Sets and Systems: Theory and Applications*, Academic Press.
[17] Zimmermann, H. J. (2001). *Fuzzy Set Theory and Its Applications*, 4th ed., Kluwer.
[18] Lee, C. C. (1990). Fuzzy logic in control systems: fuzzy logic controller – part I. *IEEE Trans. Syst., Man. Cybern.*, **20**(2), 404–418.
[19] Lee, C. C. (1990). Fuzzy logic in control systems: fuzzy logic controller – part II. *IEEE Trans. Syst., Man. Cybern.*, **20**(2), 419–435.
[20] Jang, J. S. R., Sun, C. T. and Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall.
[21] Kohonen, T. (1989). *Self-Organization and Associative Memory*, 3rd ed., Springer-Verlag.
[22] Patyra, M. J., Granter, J. L. and Koster, K. (1996). Digital fuzzy logic controller: design and implementation. *IEEE Trans. Fuzzy Systems*, **4**(4), 439–459.