Real-Time Fuzzy Processor on a DSP

Enrique Frías-Martínez
[1] Dpto. de Tecnología Fotónica
ETSI Telecomunicación, Universidad Politécnica de Madrid
28040 Ciudad Universitaria s/n, Madrid, Spain
e-mail: efrias@tfo.upm.es

Abstract - **Fuzzy logic has been successfully introduced in control applications, but usually has the problem of low speed. Fuzzy logic applications can be seen as any other signal processing application, which, taking also into account that DSPs are classical platforms in industrial environments, makes a DSP a good candidate for high speed fuzzy processing. This paper presents a real-time full-programmable fuzzy processor using piecewise-linear interpolation techniques and implements it using a DSP. A full-programmable fuzzy processor is defined as a system where the T-norm, T-conorm, aggregation operator, propagation operator, rules, membership functions and defuzzification algorithm can be defined by any valid algorithm. Real-Time fuzzy processing is defined as processing the knowledge base in a constant time and with a minimum speed of 1 MFLIPS.**

1. INTRODUCTION

DSPs are widely used in any kind of industrial application which needs high speed processing for signal processing applications.
Fuzzy Logic techniques are also used in industrial applications, as nuclear plant supervision [9], medical applications [10] or automotive applications[12]. In [11] and [1] a wide variety of fuzzy logic applications are presented. Nevertheless the main drawback of fuzzy logic solutions ha been the processing speed of a fuzzy knowledge base. Applications solved using fuzzy logic techniques can be considered as any other signal processing application. This allows to consider DSPs as a good platform to execute fuzzy logic solutions.
[13] and [14] are examples of fuzzy logic applications implemented on a DSP.
The problem of using a standard architecture, as a DSP, to process a fuzzy logic system is the final speed obtained. In this paper we present a compiler that allows to obtain real-time capabilities in fuzzy logic processing when using a DSP.

II. FUZZY PROCESSING: STATE OF THE ART

The need to process fuzzy knowledge base systems with high speed resulted in the development of fuzzy hardware architectures. This first developments were done in the mid-80's by Togai [5] using a digital architecture, and by Yamakawa [6] using analog techniques. Before that, other ASICS designed to process fuzzy knowledge bases with high speed were developed [4][2][8].

The problems of using ASICs are the final cost of the platform, and that the final solution needs a host, which will take care of the non-fuzzy logic processing. Those problems can be solved using a standard architecture, as a DSP. The reasons for using DSPs for high-speed fuzzy processing are:

- DSPs have reached a point where their speed is fast enough to process fuzzy systems with high speed.

- The use of a compiler that adapts a fuzzy system to a DSP architecture allows to obtain high-speed.

The key concept is given by the second reason. The fuzzy syntax of the rules is a very useful way to represent knowledge, and the fuzzy algorithm is a very useful way of processing it, but, it is not the best way to process it with a standard architecture. So, in order to process fuzzy systems with high speed with standard architectures, it is needed an off-line processing, a compilation, from the fuzzy system syntax to a syntax suitable for a standard architecture. This compilation is usually done using the surface control of the fuzzy system because can be obtained with mathematical tools which can be executed very fast in a standard architecture.
There are compilers developed mainly for Takagi-Sugeno (TSK) systems, and are usually based on interpolation [3][4].
A characteristic common to all fuzzy coprocessors is that only the set of rules and the membership functions of the system can be defined, because the fuzzy algorithm is implemented by hardware, and can not be programmed. This problem is also presented in fuzzy compilers, they are designed for a specific fuzzy system (usually TSK), and again, only rules and membership functions can be defined.
In this paper a full-programmable fuzzy system is defined as a system where the rules, membership functions, the T-norm, the T-conorm, the propagation operator, the aggregation operator, and the defuzzification algorithm can be defined. A full-programmable fuzzy compiler will allow to execute any kind of application using standard hardware.
In the rest of the paper, first a full-programmable fuzzy compiler is introduced. Then, the high level architecture of the controller is presented, and the following points present the speed achieved for different systems.

0-7803-7241-7/01/$10.00 (c)2001 IEEE

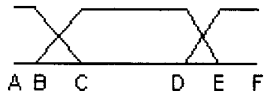
III. FULL-PROGRAMMABLE FUZZY MODEL

The model proposed is divided in an off-line processing and an on-line processing.

A. Off-line processing

FS is a vector $FS=(T_n T_c PO,AO,D,R,MF,MF_o,I,O)$ that describes a fuzzy system with T_n the T-norm, T_c the T-conorm, *PO* the propagation operator, *AO* the aggregation operator, *D* the defuzification algorithm, *R* the set of Rules, *MF* the membership functions of the input, MF_o the membership functions of the output, *I* the inputs and *O* the outputs of the system.

For each $I_i \in I$, the Activation Intervals of I_i, AI_i, are defined as the union of the set of intervals, left-closed and right-open, except the last one which is also right-closed, given by the extreme points of the kernel and the support of each one of the membership functions defined in I_i. AI_i is obtained from $(MF_{i,1},...,MF_{i,p})$ with $p=Card(MF_i)$. Fig. 1 presents an example of Activation Intervals.



$AI=\{[A,B),[B,C),[C,D),[D,E),[E,F]\}$

Fig. 1 Activation Intervals AI

The Activation Intervals can be obtained from the Activation Points. The Activation Points are defined as the set of points of each dimension of the system that define the kernel and support of each linguistic label. Formally, the Activation Points, AP_i can be obtained as:

$$AP_i = F_e(F_o(\cup(Supp_e(MF_{i,m}) \cup Kernel_e(MF_{i,m})))) \text{ with } m=1,...,p. \quad (1)$$

From that, the Activation Intervals, are obtained as:

$$AI_i = F_{in}(AP_i), \quad (2)$$

where $Supp_e$ and $Kernel_e$ are functions that obtain the extreme points that define the kernel and the support of a label, F_o is a function that orders a set of points, F_e is a function that eliminates the repeated points of a set, and F_{in} obtains the intervals, left-closed and right-open, that are defined by a set of points.

Given AI_i, $i=1,...,Card(I)$, and *S* the n-dimensional input space given by $(I_1,...,I_n)$, *P* is defined as a partition of *S* given by the cartesian product (×) of AI_i, $i=1,...,Card(I)$:

$$P = \times AI_i,\ i=1,...,Card(I). \quad (3)$$

The partition *P* is defined in a way that separates zones where the designer has a complete certainty of the output from the zones where the certainty is only partial.

This is done by defining *P* from *AI*, because the *AI*, for each dimension, separate the kernels of each label, the zone where the designer has a complete certainty, from the transition of the labels, where the certainty is only partial.

Example:

$FS=(T_n T_c PO,AO,D,R,MF,MF_o,I,O)$ is a fuzzy system with the membership functions defined as seen in *Fig. 2*. The set of *MF* defined are $\{A_{11},A_{12},A_{13}\}$ and $\{A_{21},A_{22},A_{23}\}$, defined respectively in I_1 and I_2. AI_1 and AI_2 will be (*Fig. 2*):

$AI_1=\{[0,A),[A,B),[B,C),[C,D),[D,E]\}$,
$AI_2=\{[0,A'),[A',B'),[B',C'),[C',D')[D',E']\}$.

P can be calculated with AI_1 and AI_2 (*Fig. 2*):

$P=\{\{[0,A),[0,A')\},\{[0,A),[A',B')\},\{[0,A),[B',C')\}...\}$.

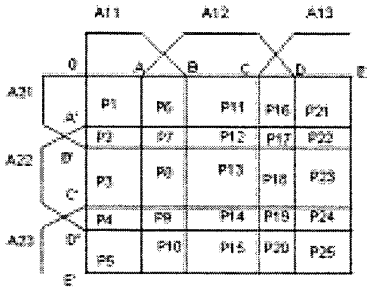


Fig.2: Representation of *MF*, *AI and P* of *FS*

■

The model is based on the value of the fuzzy system *FS* in the set of vertex that define *P*. For that, the matrix *V* is defined as a matrix that contains the set of vertex of partition *P*. *V* can be obtained from the Activation Points as:

$$V = \times AP_i,\ i=1,...,Card(I). \quad (4)$$

The Characteristic Matrix *CM* contains the value of the fuzzy system *FS* in each vertex of the partition *P*. *CM* is defined as:

$$CM=FS(V), \quad (5)$$

where *FS(V)* represents the value of each element of matrix *V* in the fuzzy system given by $FS=(T_n T_c PO,AO,D,R,MF,MF_o,I,O)$.

Example:

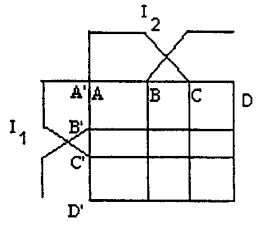


Fig. 3 Partition *P* of the fuzzy system *FS*

Given *FS* a bidimensional fuzzy system (Fig. 3), and *P* the partition of the input space, *V* and *CM* can be obtained as:

$$V = \begin{pmatrix} (A',A) & (A',B) & (A',C) & (A',D) \\ (B',A) & (B',B) & (B',C) & (B',D) \\ (C',A) & (C',B) & (C',C) & (C',D) \\ (D',A) & (D',B) & (D',C) & (D',D) \end{pmatrix}$$

$$CM = \begin{pmatrix} SB(A',A) & SB(A',B) & SB(A'.C) & SB(A',D) \\ SB(B',A) & SB(B',B) & SB(B',C) & SB(B',D) \\ SB(C',A) & SB(C',B) & SB(C',C) & SB(C',D) \\ SB(D',A) & SB(D',B) & SB(D',C) & SB(D',D) \end{pmatrix}$$

■

B. Equalization and Normalization of the inputs

The compiler also defines a set of equalization and normalization functions.
An equalization function, $E_k(I_k)$, is defined for each input of the system as:

$$E_k(I_k) = \begin{cases} 0 \text{ if } I_k \in AI_{k,0} \\ 1 \text{ if } I_k \in AI_{k,1} \\ \dots \\ p-1 \text{ if } I_k \in AI_{k,p-1} \end{cases} \quad (6)$$

with $k=1...Card(I)$ and $p=Card(IA_k)$. The equalization function is defined to identify in which *Activation Interval* is the input included. The set of values of the equalization functions of a system allow to identify the active cell, the cell in which the input is included.

A normalization function, $N_{k,r}(I_k)$, is defined for each Activation Interval of each dimension *k*. The normalization function normalizes between 0 and 1 the actual input in the active cell. The normalization function $N_{k,r}(I_k)$ is defined as:

$$N_{k,r} = \frac{1}{(B-A)}(I_k - A) \quad (7)$$

Fig. 4 shows the normalization done by $N_{k,r}(I_k)$.

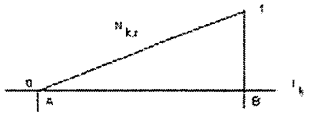


Fig.4 Normalization done by $N_{k,r}(I_k)$.

The proposed model defines a compilation of the programmable fuzzy system *FS* to a specification of the same system (Characteristic Matrix, Equalization functions and Normalization functions) suitable to be executed in a standard architecture.

C. On-line processing

The on-line processing is divided in two steps:

• The first step obtains the relevant information to obtain the output of the system. First of all, the equalization of each dimension of the system is obtained as:

$$a_1 = E_1(I_1), \dots, a_n = E_n(I_n). \quad (8)$$

The vector $(a_1,...,a_n)$ identifies the cell of *P* in which the input is included. The Characteristic Vector of an input, *CV(I)*, is defined as the set of 2^n output values of the original fuzzy systems in the set of vertex that define the cell in which the input is included. *CV(I)* can be obtained from the Characteristic Matrix, *CM*, as:

$$CV(I) = \{CM_{a_1,a_2,\dots,a_n}, CM_{a_1+1,a_2,\dots,a_n}, \dots, CM_{a_1,a_2,\dots,a_n+1}\}$$

In this step, the normalization of the input in the active cell, *N(I)*, is also obtained as:

$$N(I) = (N_{1,a_1}(I_1),\dots,N_{n,a_n}(I_n)) \quad (10)$$

• The second step, using the information given by the Characteristic Vector, *CV(I)*, and *N(I)*, calculates the output of the system *O*. The output of the fuzzy model *O*, will be calculated with a function F_o of *CV(I)* and *N(I)*:

$$O = F_o(VC(I),N(I)). \quad (11)$$

F_o has to produce a value similar to the output produced by the fuzzy system *FS*, and it has to be evaluated with high speed. The F_o proposed for the model is a multilinear interpolation among the values of *VC(I)* using *N(I)*.

IV. ARCHITECTURE OF THE CONTROLLER

The controller implemented is divided in four modules, the sensor, the interpolation cache, the model memory and the inference engine, interconnected as seen in Fig. 5.

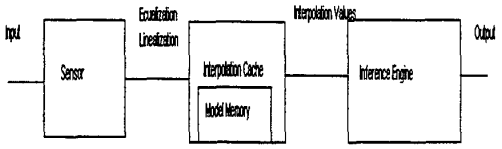


Fig. 5. Interconetion of the modules of the Controller

A. Sensor

The sensor implements the set of Equalization and Normalization functions, $E_k(I_k)$, $N_{k,p}(I_k)$. Typically in a control systems this functions can de discretized, so in execution time there is no need to evaluate them.
From the input of the system *I*, the sensor obtains $(a_1,...,a_n)$ and *N(I)*.

B. Model Memory

The Model Memory stores the Characteristic Matrix of the system, *CM*.

C. Interpolation Cache

The Interpolation Cache receives the equalization and the normalization of the input from the sensor. From the equalization of the input, it obtains the Characteristic Vector *CV*.
The interpolation cache receives the equalization of the inputs, and compares if the present input is in the same cell as the previous one. If the input is in the same cell, the actual Characteristic Vector *CV* will be equal to the previous one, so there is no need to access the Model Memory, and, if not, the Interpolation Cache accesses the Model Memory and obtains the new *CV*.
Due to the locality of the inputs, if an input is in a cell P_k, the most possible situation is that the next input of the system *I*, will be in the same cell P_k. This means that the Inference Engine will have to work with the same *CV* as the one used in the previous inference, so there is no need to access again the Model Memory to obtain it.
The output of the Interpolation Cache is the Characteristic Vector *CV* and the normalization of the input.

D. Inference Engine

The inference engine receives the Characteristic Vector *CV* and the normalization of the input *N(I)* and obtains the output *O* applying multilinear interpolation.
For a N-dimensional input system, the number of interpolations will be $2^N - 1$. The number of interpolations with the first dimension will be 2^{N-1}. For the second dimension the number of interpolations will be 2^{N-2}, and so on. These interpolations have dependencies among them.

The interpolations related with the variable *k*, can not be done until all the interpolations related with the variable *k-1* have been done.
The code that implements the Inference engine is very simple, and consist on doing the necessary number of interpolations. And example of the code of a 2 input / 1 output inference engine in C is:

```
Tmp0=CV[0]+N[0]*(CV[1]-CV[0]);
Tmp1=CV[2]+N[0]*(CV[3]-CV[2]);
Output=Tmp0+N[1]*(Tmp1-Tmp0);
```

Where *N[0]* and *N[1]* are the normalized values of the two dimensions of the system, and *CV* is the structure that implements the *Characteristic Vector*. The output of the system is stored in *Output*. Although this code could have been implemented using a loop, it has been unrolled in order to achieve higher speed when executing the code. This proves that the model compiles the knowledge an adapts it to be executed in an efficient way in a standard architecture.

V. OUTPUT COMPARISSON OF THE SYSTEM

This point gives a comparisson between the output of *FS* and the output obtained with the proposed model.
The comparison is made for a 2 Input / 1 Output system with Max-Min as inference system and COG as deffuzzification algorithm. The results are obtained from comparing 48682 values of the two systems. The results are summarized in *Table 1*.

Table 1. Results of the Comparisson of the two models.

Values without Error	35 %
Values with Error < 10 units	65 %
Average of the Error	11 units

The range of the output of the system is 1900 units, and an average error of 11 units makes that the error of the output is in average 0.77% over the range of the output.

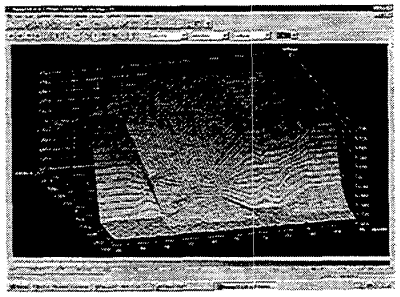


Fig. 6 Surface Control obtained with the model.

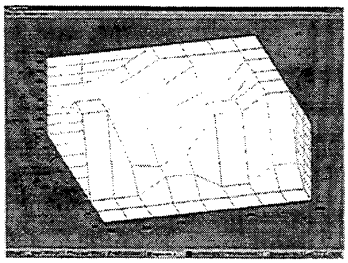


Fig. 7 Surface Control obtained with the Fuzzy System *S*.

Because the special characteristics of the design of a fuzzy system, the important thing is not the concrete value for a given point, but the general behavior of the system. This comparison can be made using the surface control given by both controllers in Fig 6 and Fig. 7.
The surface control of the model has been obtained with Matlab, and the one of the fuzzy system *FS* with FuzzyTECH[1].
The conclusion is that the model gives the same behavior as the fuzzy system, but with less information. This is because the compiler is designed to keep the certainty of the designer. This is done by defining the partition *P* from the *Activation Intervals*, because the *Activation Intervals*, for each dimension, separate the zones of certainty of the designer, the kernel of the label, from the zones where the certainty is only partial, the transition between 0 and 1 of the labels.

VI. SPEED COMPARISSON OF THE SYSTEM

The system has been implemented in a Texas Instruments (TI) DSP, TMS320C6201[7]. The implementation has been done in a DSP because is the typical architecture for signal processing, and because it has been designed for high speed processing.
The code has been developed in C, and has been optimized successfully using TI compiler options. Although the code has been developed in a high level language the speed achieved is very satisfactory. This has an important advantage, the inference engine developed can be executed in any architecture with C support.
The speed has been tested with a 2 input / 1 Output system, a 3 Input / 1 Output system and a 4 Input / 1 Output system. The results obtained can be seen in *Table 2*.

Table 2. Execution Time of the model on a C6201

System	Clock Cycles	Time (ns)	MFLIPS
2 I / 1 O	44	220 ns	4.5
3 I / 1 O	106	512 ns	1.8
4 I / 1 O	211	1055 ns	0.95

The results given in *Table 2* are completely satisfactory, and can be compared with the results obtained with others architectures. *Table 3* gives the time response of some fuzzy coprocessors to compare the speed obtained.

Table 3.Processing Time of some Fuzzy Coprocessors

System	FZP-0401A	WARP 2.0	SAE81C99
2 I / 1 O	0.48 MFLIPS	----------	----------
3 I / 1 O	----------	----------	12 µs
4 I / 2 O	----------	33.1 µs	----------

FZP-0401A[4] is an ASIC that processes fuzzy systems also using interpolation but only for Takagi Sugeno system. WARP 2.0[8] and SAE81C99[2] are fuzzy commercial coprocessors of ST Microelectronics and Siemens respectively.
The conclusion is that standard architectures can process fuzzy knowledge even faster than specialized coprocessors using a compilation that adapts the knowledge to the standard architecture.

VII. CONCLUSIONS AND FUTURE WORK

This paper has presented a compiler that allows to execute real-time full-programmable fuzzy systems using a DSP. This is extremely useful because fuzzy logic is being introduced in more and more industrial applications, which usually require high speed, and because a DSP is a typical platform in industrial environments. Using a standard architecture as a DSP allows to develop the final solution very fast and at a low cost, compared to specific fuzzy circuits. Also, a full-programmable model allows to use the same hardware and the same model with any kind of application
The final speed obtained can be improved using the new elements of the family, for example, the last development of the C62 family, the C6203 has a 3.3 ns clock (300 MHz.), which will allow to multiply the speed obtained by 3/2.
The compiler proposed does not depend on any special architecture an can be executed in other DSPs, or other architectures like MCU and microcoprocessorss using the same code.

VIII. ACKNOWLEDGMENTS

This paper is supported by grant CICYT-TIC96-1393-C06-01.

IX. REFERENCES

[1] C. Von Altrock, *Fuzzy Logic&Neurofuzzy Applications in Busines*, Prentice-Hall 1997

[2] H. Eichfeld, "A 12b General-Purpose Fuzzy Logic Controller Chip", *IEEE Transactions on Fuzzy Systems*, Vol 4, No. 4:460-475 ,1996

[3] R. Rovatti, "Linear and Fuzzy Piecewise-Linear Signal Processing with an Extended DSP Architecture", in *Proceedings of the 7th IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 98*, pp. 1082-1087

[4] N. Yubazaki, "Fuzzy inference chip ZP-0401 based on interpolation", *Fuzzy Sets and Systems*, Vol.98(3):299-310

[5] M. Togai, "Expert System on A Chip: An Engine for Approximate Reasoning", *IEEE Expert*, Vol. 1, No. 3:55-62, 1986

[6] T. Yamakawa, "A simple fuzzy computer hardware system employing MIN&MAX operations", in *Proceedings of the 2nd IFSA Congress*, pp. 122-130, 1987

[7] Texas Instruments, "TMS320C62x/C67x CPU and Instruction Set. Reference Guide - SPRU 189C", March 1998

[8] SGS-Thomson Microelectronics, "Weight Associative Rule Processor WARP 2.0" ,1994

[9] J. Gebhardt and C. Von Altrock, "Recent Successful Fuzzy Logic Applications in Industrial Automation", in *Proceedings of the 5th IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 96*, New Orleans, Sep. 1996

[10] G. Zahlman and M. Scherf, "Monitoring Glaucoma by Means of a NeuroFuzzy Classifier" in *Fuzzy Logic Aplication Note series*, Inform Corp, http://www.fuzzytech.com, 1998

[11] M. Jamshidi, A. Titli, L.A. Zadeh, S. Boverie, *Applications of Fuzzy Logic. Towards a High Machine Intelligence Quotient Systems*, Prentice-Hall, 1997

[12] D. Elting, M. Fennich, R. Kowalczyk, B Hellenthal, "Fuzzy Anti-Lock Brake System Solution" in *Intel Corporation's Automotive Operation Center Reports*, http://developer.intel.com/design/mcs96/designex/2351.htm, 1998

[13] B. Bierke and C. Von Altrock, "Fuzzy Logic Enhanced Control of an AC Induction Motor with DSP", in *Proceedings of the 5th IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 96*, New Orleans, Sep. 1996

[14] Texas Instruments, "Fuzzy Logic: An Overview of the latest Control Methodology" in *TI Application Report*, 1993