CHAPTER 3

Applications to High Energy Physics Experiments of a High Rate 2-Input VLSI Fuzzy Processor

3.1. Introduction

Although fuzzy logic is currently mainly applied to control systems [3], we have investigated its applications in other fields such as HEPE. In particular we show how a fuzzy system can improve the performances of the front-end electronics implemented in HEPE. In this field every event that is, for example, an elementary particle collision, generates a big amount of data that has to be analyzed in real time in order to save it only in case of interesting events. For example, beside significant data, much noise may also be collected and it should be rejected in real time if possible. As far as the electronic devices used in HEPE, we call trigger the apparatus able to select or reject the events depending on their importance. Usually there are several trigger levels depending on the decision time that is the time required by an electronic device for deciding whether to store or reject the data: it usually depends on the kind of experiment.
For example, for most HEPE, the first trigger level is designed for a decision time smaller than 100 ns, while some microseconds are required for the second level and few milliseconds for the third level.
In most of these experiments the job of these trigger levels is pattern recognition one. This is due to the fact that each datum of a given event belongs to a defined data cluster. This is why we are involved in an cluster recognition problem. Since in HEPE any given area may represent or be associated to a specific particle, the problem of particle detection or particle recognition may be reduced to an area detection task.
So far, to face and solve these pattern recognition problems in HEPE, rigid electronics hardware devices have been applied while we have been looking for new, more flexible and reconfigurable architectures. Although interesting investigations have been carried out in the last years about fuzzy hardware solutions [14], we have finally decided for HEPE purposes to design a dedicated fuzzy architecture both to meet the high-speed constraints and to provide more flexibility within experiment requirements. So we have firstly evaluated the performances of a fuzzy system to detect a given area and then we have designed a fast digital fuzzy processor able to do this job, within the time required by the first or second trigger level: in fact these are the levels that require a more sophisticated electronics. We have designed and realized a first prototype [8] with a 1.0 μm digital technology and, from that, we have improved its performances by implementing new technologies and new fuzzy architectures [15]. Eventually, the reason why we have not used commercial general purpose fuzzy processors [16], [17] relies on the fact that their computation speed is not as high as required in HEPE [5], [18] since most of them have been designed for controls.
This work deals with the description of how the fuzzy system works for applications to area recognition. Three simple examples are given: the first is a three rectangular shape area differentiation while the second and the third are respectively a normal and a

rotated convex area detection within a rectangular shape background. As shown in the figures 3.1., 3.2. and 3.3., the system reads two inputs by means of integer variables within a given input domain, and calculates the fuzzy system output, that represents which of the groups the input data pair belongs to.
In the subsection 3.7. is explained the functional principle of the commercial Adaptive Fuzzy Modeler (AFM) Software (SW) [19] we have used for generating the fuzzy systems. Since it is a commercial one we do not describe it in details except for what regards its features and performances. What we emphasize is the application for our fuzzy processor. In fact it is one of the neural network fuzzy rule generators that may be applied for problems like ours in physics experiments [6].

3.2. Pattern Recognition

Since the knowledge base needed to solve a given problem is not always available, it is useful to extrapolate it from input-output patterns related to the problem. In this case it proves to be very useful to use a neural network able to learn starting from the I/O patterns. For the automatic generation of fuzzy systems we used the Adaptive Fuzzy Modeler (AFM) SW, a neural network based software developed by SGS-Thomson.
We remind that a fuzzy system consists of all the parameters that identify a given fuzzy algorithm so that, once an input data set is read, the output value is carried out. In details, the fuzzy systems include the number of input and output variables, the number and the shape of the membership functions associated to each variable, the fuzzification and defuzzification methods and the fuzzy rules.
The AFM software asks the user to provide as inputs the number of input and output variables and the number of membership functions required by any variable. Then the input-output patterns teach the neural network that returns as output the features of every membership function required for the input variables.
The total processing time of a fuzzy system usually depends on the number of fuzzy sets per input variable and consequently on the number of fuzzy rules that make up the overall knowledge base of the problem. For this reason the processing time would increase a lot if we used 8 fuzzy sets per input variable instead of 3 ones. In order to avoid this kind of problem we have designed a fuzzy architecture whose total processing time in independent of the number of fuzzy sets and rules.
For the learning stage, we have chosen three membership functions for each variable and 1000 input/output patterns. This stage takes nearly 600 loops to find the best fuzzy system in order to recognize most of the examples. After generalization, that is the test phase with new examples, we have proved that the fuzzy system recognizes up to 99% of the input patterns. So far the AFM software generates a fuzzy system able to recognize the areas shown in the examples, and, consequently, it is ready to be implemented into the fuzzy chip. Thus, the same task that is made by software fuzzy simulators for example on personal computers may be done in hardware in a very short time (hardware accelerators).
All in all, the examples here presented made just the first stage of our investigation in pattern recognition problems using fuzzy logic.
This research is still going ahead and we are using areas of different shapes. As far as area rotations, Figure 3.3. shows a more complex problem by means of a convex shape rotated anti-clockwise by an angle of 10 degrees. The problem of rotated shapes may

not be solved by traditional algorithms based on, for example, digital comparators. In fact, if the problem is just to recognize areas delimited by horizontal and vertical lines like in figures 3.1. and 3.2., very simple digital electronics may do this.
On the other hand, if the same shapes occur rotated with respect to how the electronic has been designed, as shown in Figure 3.3., a rigid device may no longer solve the problem in such a short time, while a fuzzy system seems to be more flexible and fast due to the fact that fuzzy logic implies degrees of truth instead of digital true-false relationships. Moreover each fuzzy rule has a "local effect" in the sense that is related to a given subset of the input variable ranges. In this way it is possible to tune the fuzzy system to take into account of a local pattern modification suggested by new experimental data. Consequently the rotated shapes that are not so easy to recognize using traditional electronics, give acceptable errors using fuzzy logic.
For example in Figure 3.2., in case not rotated, the 99% of input patterns are recognized while, in case of the same figure after being rotated, are recognized just the 95% of the patterns but we outline that the decision time is the same.

3.2.1. Pattern Recognition: Three Areas Problem

In this case we have three polygonal areas such as A, B and C as shown in Figure 3.1. In some cases of pattern recognition problems, it is in use to associate to each output pattern a crisp value. In our case it has been used an output integer identification variable that may assume values that belongs to {-2;0;2}.
The job is to detect if a given pair of (x,y) input variables belong to the set A or to the set B or to the set C. Let us define the three sets A, B and C:

$A=\{(x,y):(x\in[5,7]\wedge y\in[5,14])\vee(x\in]7,12]\wedge y\in[12,14])\}$

$B=\{(x,y):(x\in]7,9]\wedge y\in[5,12])\wedge(x\in]9,12]\wedge y\in[9,12]\}$

$C=\{(x,y):(x\in]9,12]\wedge y\in[5,9[)\}$

with $5<=x<=12$ and $5<=y<=14$.

In other words we seek for a function *f* such that:

$f:\{A\vee B\vee C\}\rightarrow\{-2;0;2\}$

So if the result is 2 this means that the input pair (x,y) belongs to A while when is 0 it belongs to B and belongs to C if the result is -2.

3.2.2. Pattern Recognition: Normal Convex Areas

In this example we have a polygonal area A included within a rectangular background B such as shown in Figure 3.2. The job is again to detect if a given pair of (x,y) input variables belongs to the set A or to the set B but in this case, as previously mentioned, the figure may occur rotated of a given angle.
Nevertheless, let us define the two sets A and B:

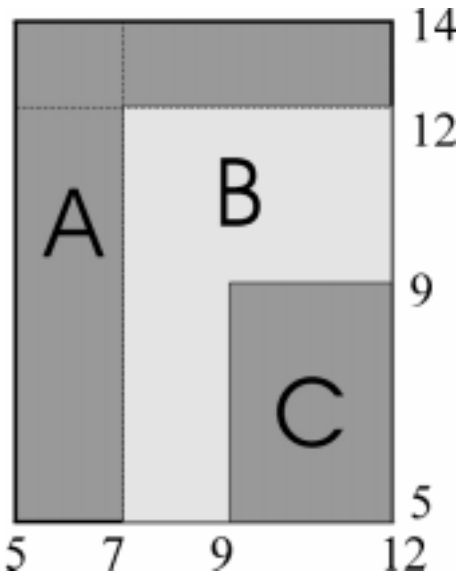

Figure 3.1. Three Rectangular Shapes

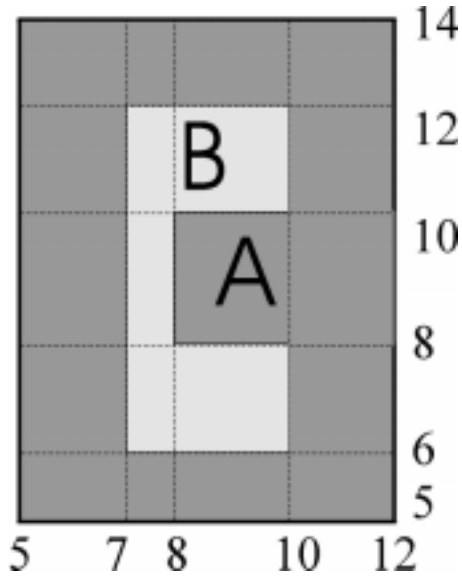

Figure 3.2. Normal Convex Area

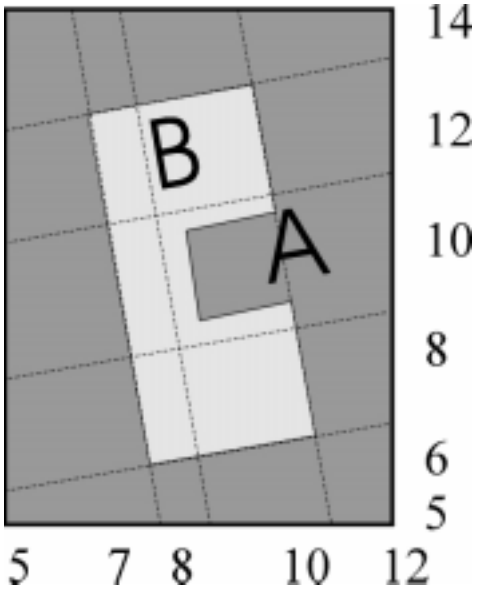

Figure 3.3. Rotated Convex Area

A={(x,y):(x∈[5,7]∧ y∈ [5,14])∨(x∈ [10,12]∧ y∈ [5,14])∨(x∈ [8,10[∧y∈ [8,10])∨
(x∈]7,10[∧y∈[5,6])∨(x∈]7,10[∧y∈ [12,14])}

B={(x,y):(x∈]7,8[∧y∈]6,12[)∨(x∈]8,10[∧y∈]6,8[)∨(x∈]8,10[∧y∈]10,12[)}

with 5<=x<=12 and 5<=y<=14.

The function research in this case is similar to the case of Figure 3.1. except for the fact that, having two areas A and B to differentiate, there would be just two output crisp values:

$f:\{A \vee B\} \rightarrow \{-2;2\}$

3.2.3. Pattern Recognition: Rotated Convex Areas

In this example of Figure 3.3. we again have a polygonal area A included within a rectangular background B but now the figure is rotated 10 degrees anti-clockwise. The job is again to detect if a given pair of (x,y) input variables belongs to the set A or to the set B.

3.3. Two Input Fuzzy Processor

In order to be able to design and realize a fuzzy processor useful for HEPE and according to the AFM features, our first design step was to force the AFM SW into using a given fuzzy system configuration that could be quite easily implemented on hardware later. So far we decided to design a chip that allows to use up to 8 fuzzy sets for each input variable and an overlap between adjacent fuzzy sets not greater than 2.
8 fuzzy sets have been chosen even if 3 ones have proven to be sufficient in the examples given of pattern recognition: this has been done to give more flexibility to the system, so that it can be used even in the case of more complex areas recognition. Besides limitation on the overlapping(max 2) allows us to design an architecture whose decision time does not change with the number of fuzzy sets, as described later in the section.
With these features the number of input fuzzy set combinations is $8^2=64$ as shown in Figure 3.4. For this reason all the possible fuzzy rules are 64 even if the number of active rules, which can give a non-null contribution, is reduced to $2^2=4$.
An other key point concerns the rule inference process. We have decided to use fuzzy rules that imply only the “AND” conjunction implemented with a minimum operation (see section 3.1 for details). Finally, we have faced the problem of choosing the inference and defuzzification method. As will be described in section 3.5, we have implemented the Sugeno order 0 methods [20], [21]. Of course we have chosen this method since it gives good results in area recognition problems and allows both an easy hardware implementation and a fast execution time.
All of these features have been the main specifications from which the design of the fuzzy processor took place. The main features of the fuzzy processor in the final version are summarized in Table 3.1. Let us now start describing in details the fuzzy processor blocks.

3.3.1. Two Input Fuzzy Processor: Main Blocks

Follows a list of the main logic blocks that compose the fuzzy processor and a brief description that refers to the flow chart in Figure 3.5. For each bus the number of bits is specified.

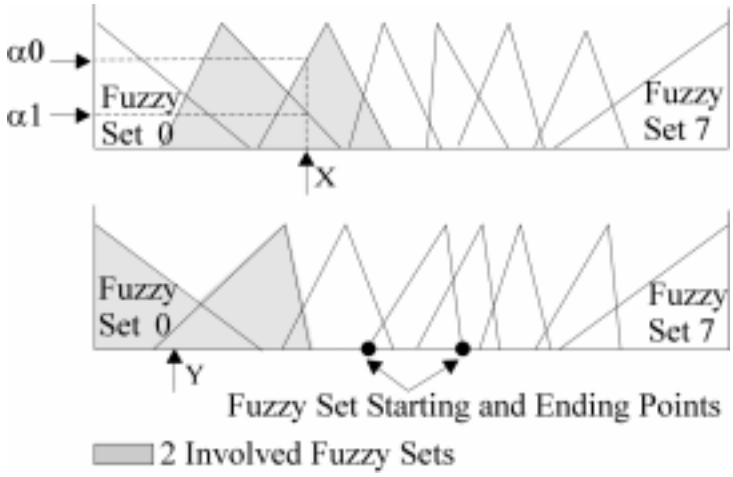

Figure 3.4. Fuzzy Set Distribution

- The *active-interval-selector* compares the input variables to the points stored into the *membership-functions-interval-memory*, and provides an interval 3-bit code to the *address-generator*.
- The *membership-function-interval-memory* (look-up-table) stores the starting and ending points of the fuzzy sets.
- The *address-generator* generates the addresses for the *rule-memory* starting from the X and Y interval codes generated by the *active-interval-selector*.
- The *rule-memory* contains all the rules of the fuzzy system.
- The X and Y *membership-function-memory* blocks (look-up-tables) store the input membership function shapes and together compose the *fuzzifier*.
- The *minimum-operator* implements the fuzzy *and* conjunction and extracts the minimum value, here in called Θ, between the two αs, that represents the rule premise degree of truth.
- The *inference & defuzzifier* is the circuit that computes the output result: it is composed of two *adders*, a *multiplier* and a *divider*. Θ*Z is carried out rule by rule and added to the previous partial sums ∑Θ*Z while Θ is added to the previous ∑Θ. Thus the division process ∑Θ*Z/∑Θ takes place.

3.3.2. Two Input Fuzzy Processor: The Rule Memory

The *rule-memory* is dimensioned as to contain all the possible rules that is 64 words. The rules are loaded in the *rule-memory* starting from the one that involves all the lowest fuzzy sets for the two input variables up to the one that involves all the highest corresponding fuzzy sets. In this way the address word identifies the premise fuzzy sets involved by each fuzzy rule. Thus the *rule-memory* can be organized as words of 9 bits, where a word contains only the Consequent Rule Code Z, a 7-bit word representing the output membership function crisp value according to Sugeno order 0 inference method, and the Rule Premise Code, a 2-bit word that tells which variables are present for every rule.

For example when the rule premise code is 11 both X and Y are present, when 10 only X is present and when 00 neither X nor Y are present.

TABLE 3.1. Fuzzy Processor Main Features

Inputs	2-digital 7-bit variables
Outputs	1-digital 7-bit variable
Degrees of Truth	4-bit
Fuzzy Set Overlapping	Adjacent fuzzy sets may overlap 2 at a time
Input Fuzzy Sets	8 any shape for X and Y
Output Fuzzy Sets	128 crisp values
Fuzzy Rules	64 9-bit words
Conjunction 'and' Operator	Minimum
Inference & Defuzzification Method	Sugeno order 0
Clock Frequency	up to 50 MHz
Input Rate	80 ns
I/O delay time	270 ns
technology	ES2 .7 μm CMOS digital
Current on VDD	100 mA at 50 MHz
Power Consumption	500 mW at 50 MHz
Package	DIL 48 Plastic
Silicon Area	14 mm^2

3.3.3. Two Input Fuzzy Processor: Rule Memory Address Generation

The *active-interval-selector* is a circuit that compares the input variable values to the interval points stored into the *membership-functions-interval-memory* (see Figure 3.4.). Then it sends the two 3-bits interval codes to the *address-generator*. This circuit, once got these pair of 3-bit codes, generates the four addresses that derive from the permutations of the four involved fuzzy sets that select only the active rules. For example, in Figure 3.4. the four involved fuzzy sets are the fuzzy set 1 and 2 for X variable and fuzzy sets 0 and 1 for Y variable. Consequently the four permutation codes would be: 001-000, 001-001, 010-000, 010-001. These addresses fire exactly those fuzzy rules among the 64 ones that include the involved fuzzy sets. These four fuzzy rules would be some like the following:

if (X is '001') and (Y is '000') then (Z is Zh);
if (X is '001') and (Y is '001') then (Z is Zi);
if (X is '010') and (Y is '000') then (Z is Zj);
if (X is '010') and (Y is '001') then (Z is Zk).

The indexes *h, i, j, k,* stand for an integer number from 0 to 63. In fact, since there are 64 fuzzy rules, each one may have its own Z that corresponds to a singleton output fuzzy set. This is according to the Sugeno order 0 defuzzification method. In this way

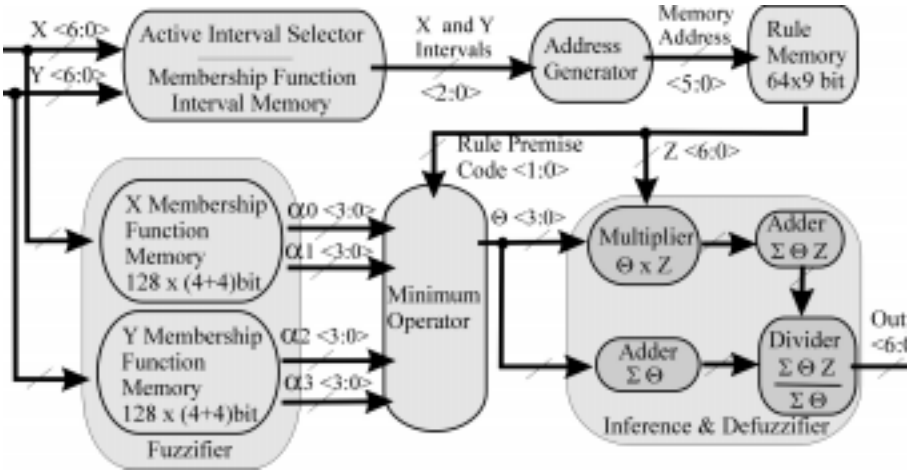

Figure 3.5. Flow Chart

the use of the *active-interval-selector* reduces the number of rules to be processed by a factor 16 from 64 to 4 without time consuming.

3.3.4. Two Input Fuzzy Processor: Fuzzification Process

The input variables X and Y select the active fuzzy sets. Thus, by addressing directly the X and Y *membership-functions-memory* blocks the four possible input degrees of truth αs are generated. In these two memory blocks 128 (4+4)-bit words are stored since the input domain is coded with a 7-bit resolution. The pairs of four bits represent the two overlapping membership function values that are the input degrees of truth related to the value of each input variable. We refer to the input degrees of truth as $\alpha 0$ and $\alpha 1$ for X and $\alpha 2$ and $\alpha 3$ for Y (see Figure 3.4.). It is to be emphasized that just two at a time are selected depending on the fuzzy rule. For example $\alpha 0$ and $\alpha 2$, $\alpha 1$ and $\alpha 2$ but obviously not $\alpha 0$ and $\alpha 1$ that are both related to X.

3.3.5. Two Input Fuzzy Processor: Premise degree of truth computation

Once the input degrees of truth αs are read from the *membership-function-memory* and selected by means of the Rule Premise Code, the rule premise degree of truth is computed. This is done by means of a *minimum-operator* block that gives as output the value of Θ.

3.3.6. Two Input Fuzzy Processor: Inference & Defuzzification Processes

When the rule inference process has been done, the multiplication Θ*Z takes place. This is the contribution of a given fuzzy rule to the final result named Output. This process is composed of two additions and a division operation. The two additions, which concern a numerator and denominator of the weighted sum final result shown in formula 1, are carried out by adding the Θ*Z and Θ values respectively to the previous partial sums $\sum\Theta$*Z and $\sum\Theta$. Finally the division process $\sum\Theta$*Z/$\sum\Theta$ can start. It should be noted that only the last division process is off pipeline while all the previous ones

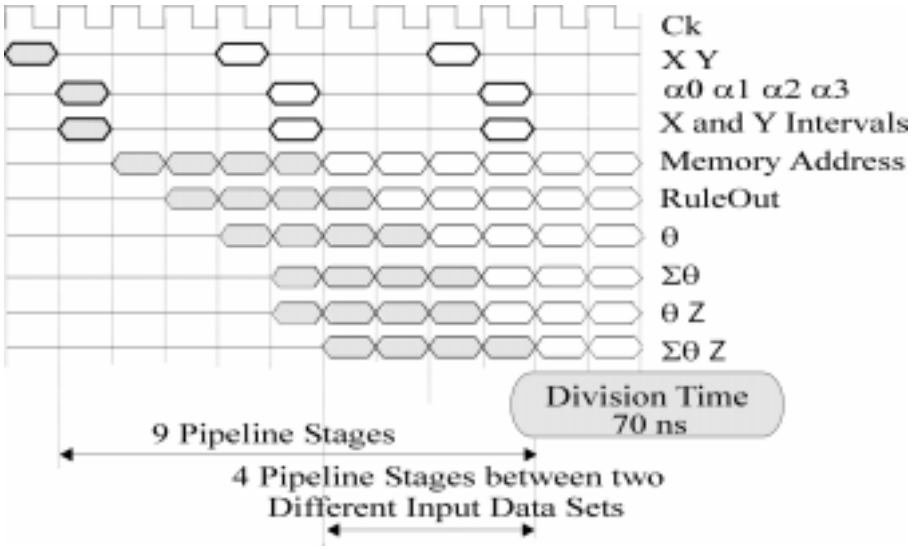

Figure 3.6. Pipeline Stages

compose the pipeline stages (see Figure 3.6.). In this way, it does not affect the rate but just delay time from input to output.
In the below formula 3.1., the two upper indexes 4 stand for the number of fuzzy active rules.

$$Output = \frac{\sum_{1}^{\#Act} Zi * \vartheta i}{\sum_{1}^{\#Act} \vartheta i} \qquad (3.1.)$$

3.4. Pipeline Subdivision

The overall architecture of the fuzzy processor is pipelined as shown in Figure 3.6., where the data flow for every pipeline stage is displayed. It is to be noted that the 9 pipeline stages shown in the figure are composed of 5 actual pipeline stages into which the fuzzy architecture has been divided and 4 pipeline stages due to the number of active rules. Nevertheless, since always 4 active rules are processed, this time can be considered as a pipeline time. So, from the moment a new data set enters the processor 9 pipeline stages are required for the fuzzification and inference processes to be executed.
In the first clock period two processes are performed in parallel: the four αs are read from the *Membership-Function-Memory* and the *Active Interval Selector* computes the X and Y interval codes. In the following period the *address-generator* produces the first address for the *rule-memory* and a period later the first rule premise codes are available for the *minimum-operator* circuit. The first Θ is produced in the 4th pipeline stage, whereas the first Θ*Z is valid one period later. After 9 periods both sums ∑Θ and

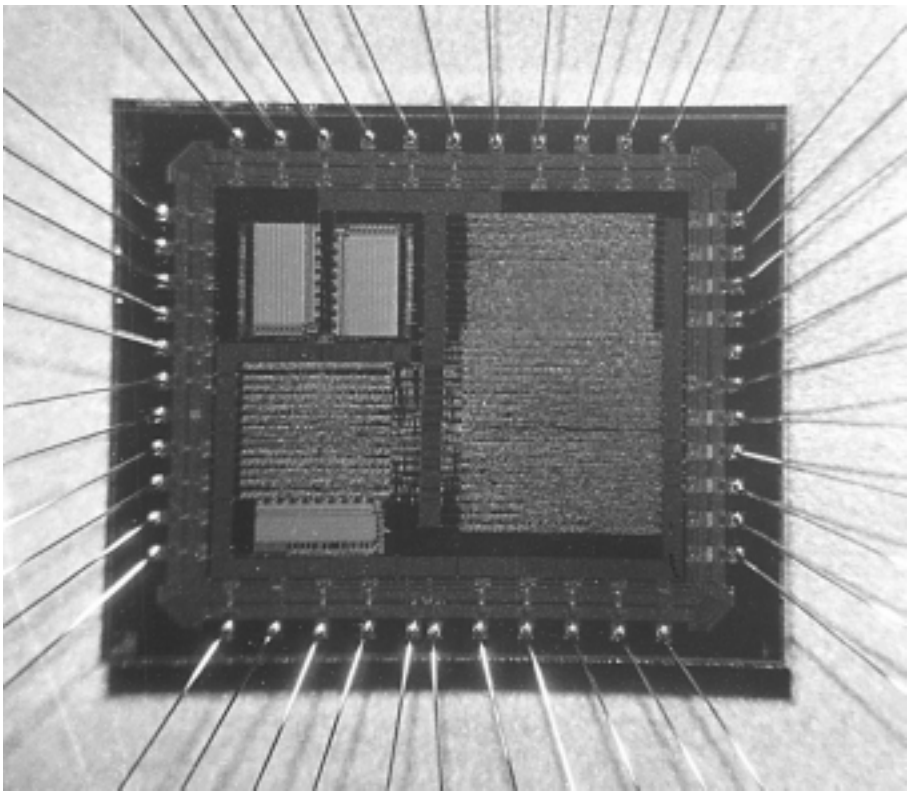

Figure 3.7. The Fuzzy Chip Microphotograph

$\sum(\Theta*Z)$ are carried out, so that the final process of division, which requires at most 70 ns, can start. What is really remarkable in this pipelined structure is that a new input data set can enter the system after only four clock periods, since at this stage all the four addresses for the *rule-memory* have already been generated. For what concerns the whole delay estimated starting from the input data set loading cycle to the corresponding output data generation, four contributions have to be considered. In fact, first the startup time due to the input synchronization, which requires one clock period of 20 ns, has to be considered; then it has to be added the time due to the actual number of pipeline stages that is 5 as above reported that is 5 x 20 ns = 100 ns; thus the time due to the number of active fuzzy rules which is 4 x 20 ns = 80 ns; eventually it has to be considered the division time that takes nearly 70 ns.
All together these delays give rise to a global processing time of 270 ns if a 50 MHz clock signal is used, but the processing time is reduced to 80 ns.

3.5. Layout Design

Figure 3.7. shows a microphotograph of the fuzzy processor that has already been realized. The silicon area is nearly 14 square mm in 0.7 μm ES2 digital technology. As can be seen, the memory blocks on the left side of Figure 3.7. do not require a large silicon area if compared to the whole layout. This justifies the look-up-table choice previously mentioned for storing the membership function shapes. In

addition, thanks to the small size layout, both the net parasitic capacitance and, consequently, the timing delays are quite small. This simplifies the layout design in terms of power supply and clock net dimensioning [7].
As far as the layout design, the processor components have been firstly divided into the logic blocks taking in consideration the logic function each component was designed to. Moreover the input/output pad distribution has been chosen as a trade-off solution in order to minimize the net connectivity. Thus, the placement and routing phases have been done after having put some constraints in terms of integrated circuit design parameters. In particular some global nets such as power supply nets VDD and GND, clock and reset nets, have been routed manually. In fact, since these nets connect hundreds of cells, it can be very dangerous let them be routed automatically: the result could be very far from what one would expect. This does not apply for short nets that connect not so many components. An other important point during routing phase concerns the net priorities. This parameter determines the wire overlapping. In fact, when two nets cross each other, the routing software has to decide which one has to be left on one single metal layer and which one must be routed on another metal layer by means of a layer contact. This point, especially for global nets such those previously mentioned, may affect significantly the connectivity and the functionality of the chip since different layers have different performances in terms of resistance, maximum current and parasitic parameters.
The clock net has been designed with a tree shaped routing style. The clock net routing has be done by means of a large net trunk from the clock pad and many small net branches from the trunk to the standard cell clock pins. This allows the clock edges to be distributed quite evenly among the cell rows and to minimize the skews. It is also important to leave a sufficient margin between the clock pad fan-out and the global net fan-in. Moreover, the standard cells that are not connected to the clock signal, or better, the logic blocks that have not to be synchronized, may stay off the clock net. In this way, the synchronous part of the design may be put together and divided into two main standard cell row blocks.
The power net width must be dimensioned taking into account the clock frequency, the number of components and their average power consumption, the estimated number of involved components for each clock cycle and the percentage of them that really commutes. This is why even if all the logic gates are always connected, the signal propagation starting from the input pads does not pass always through all of them. Moreover, where this signal propagation passes may also do not have any commutation effect.
Of course the power net dimension operation may be done only approximately and, for this reason, the net width result can be multiplied by 2 or 3 for being more confident.

3.6. Conclusions

One of the main intents of this work is to explain the possibility to apply fuzzy logic and, in particular, fuzzy processors, to other application fields beside the control and pattern recognition fields. For example here is reported the investigation that has been made in order to apply fuzzy processors to High Energy Physics Experiments. This can be done by means of a fast parallel-pipeline fuzzy architecture that, in the case of a small number of input variables, gives rise to a small size fuzzy chip. We had 10 ASIC prototypes fabricated by ES2 foundries at the end of 1997 [22]: then they have

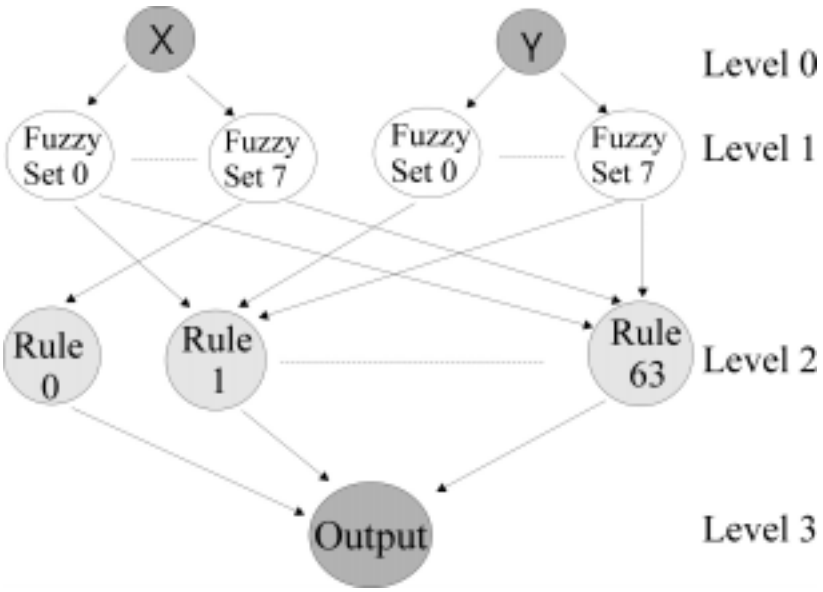

Figure 3.8. The AFM Neural Network

been exhaustively tested using the ASIC Tester LV500 by Tektronics, belonging to the Physics Department of Bologna University. It is able to run the chip up to a frequency of 50 MHz. All the chip prototypes proved to be working correctly, also thanks to the goodness of VHDL based ASIC design flow and of the SW simulators used. We feel confident on applying it for physics experiments due to its fast rate. We remind that the processor is able to compute a two input variable fuzzy system in just 80 ns for a clock signal of 50 MHz. It is also one of our future aims to design fuzzy processors that will be clocked at higher frequencies to further improve the input rate.
We finally outline that the problem of pattern recognition has been solved by using fuzzy logic obtaining a flexible and fast decision device. We are confident that this technique can be applied in several fields other than HEPE.

3.7. Fuzzy System Generation

The AFM neural network is composed of four levels as shown in Figure 3.8. The first level is made of as many neurons as the number of input variables. It is connected to the second level that is composed of as many neurons as the number of fuzzy sets of the all input variables. The connection may be done by 2-component vectors that, since the membership function shapes are triangular, represent the two slopes of the two triangle sides. These vectors are also to be related to input thresholds that represent the centers or, in other words, the positions, of the input membership functions. In addition, these thresholds bind together the second and the third neuron level. These first two neuron levels are to be taught in a first learning phase. On the other hand, in the third level of the network each neuron represent a fuzzy rule and it has binary connections to the previous second level. The connection is 0 if the related input variable is not present

in the fuzzy rule and is 1 if is present. This level is also to be taught but in a separate stage from that used for the first two levels.
The fourth level has as many neurons as the number of output variables so that, in our case, it is just one. From each fuzzy rule comes to this final neuron a connection whose weight is the output threshold specified within each fuzzy rule (what we call herein and below Z).
For the complete sketch of the AFM neural network see Figure 3.8.