Fifth IFSA World Congress (1993), 1386-1389

Architecture of a PDM VLSI Fuzzy Logic Controller with an Explicit Rule Base

Ansgar P. Ungering, K. Goser

University of Dortmund, LS-BE, Emil-Figge-Str. 68, 4600 Dortmund, Germany
Fax: + 49 231 7554450 E-mail : ungering@luzi.e-technik.uni-dortmund.de

Abstract: We are describing the architecture of a fuzzy logic controller using pulse-width-modulation (PDM) technique and a pipeline structure. Features of this controller are: A new architecture for the inference unit, reduced chip area and less I/O-pins. Additionally we present two different rule-bases: one hardwired with reduced chip-area and the other programmable for prototyping. Also an architecture of a parallel minimum-gate is shown.

Introduction

In the recent years there has been an increasing interest in methods for realizing efficient digital fuzzy controller hardware suitable for integration /1,2,3/. First implementations /1/ show a high flexibility but need a large amount of chip area. New architectures proposed in /2,3,4/ restrict the degree of overlap of the membership functions (MFs) and lead to a reduced amount of chip area. This paper presents an extension of these methods.

A typical fuzzy hardware controller consists of the following building blocks:

1. Fuzzifier unit
2. Rule-base
3. Inference unit
4. Defuzzification unit

In this communication, we will show that the first three of the building blocks mentioned above can be implemented with reduced hardware requirement but without reducing the speed of operation. This is achieved if the input and output signals are pulse-width-modulated (PDM) and if the internal operation is also based on PDM-signals. In this case the rule-base is hardwired. For prototyping we also present a programmable rule-base.

Fuzzy Operations with PDM-Signals

The described controller uses only the minimum- (MIN) and maximum- (MAX) operators. Using AND-gates and OR-gates it is very easy to calculate the MIN and MAX of PDM-signals /5,6/. The hardware requirement is very low. Since it is not very difficult to build a voltage or current to PDM converter, a fuzzy controller using PDM signals usually requires less I/O-pins and less chip area than other methods.

Analysis of the MIN/MAX-Algorithm

The algorithm is based on minimum- and maximum-operations. The "crisp" output value will be calculated using the center of gravity (COG) method. The basic principle of the wellknown algorithm is depicted in Fig. 1. The calculation of the "THEN-PART" (α-cut and combining the MFs Cx′ to one resulting output function C*) usually takes N time steps. N is equal to the x-resolution of the MFs. Therefore, if the x-resolution of the MFs is 256 (8-bit), N=256 time steps are required. This part of the controller limits the speed of operation.

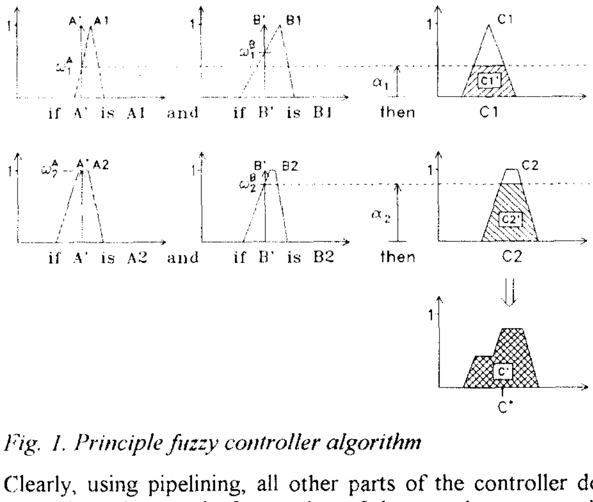


Fig. 1. Principle fuzzy controller algorithm

Clearly, using pipelining, all other parts of the controller do not reduce the speed of operation of the complete system, if their internal speed of operation does not exceed this limit.

Architecture

A Fuzzy-Controller system consist of the following blocks:

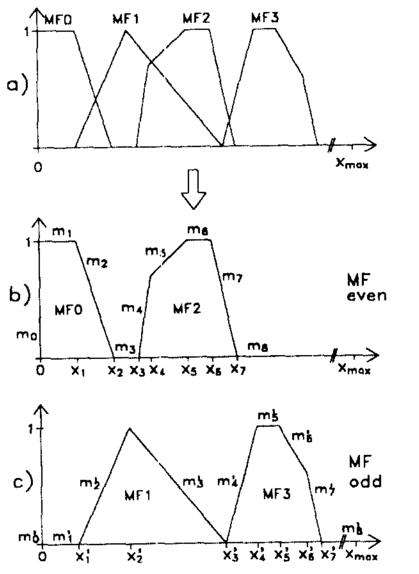
1. Fuzzifier
2. Rule base
3. Output MF-Generator
4. Defuzzifier
5. Control unit

Our architecture uses a 4-stage-pipeline which consists of the following stages: Fuzzifier, Rule base, Output-MF-Generator and Defuzzifier.
In the following paragraphs these blocks will be described. The resolution is 8-bit, so that one main clock cycle takes 256 system clock cycles. Every stage of the pipeline requires one main clock cycle.

Implementation of the Controller Blocks

1. Storing and reconstructing membership functions

It is well known, that most fuzzy applications require only a limited number of MFs with an overlap degree of 2. Therefore, we restrict the number of MFs to 8 for each input and to 8 for each output. It should be noted that these restrictions are not disturbing. They lead, however, to simple circuits. The MFs (see Fig. 2.a) are stored by the memory blocks "even" and "odd" in the following manner /3,4/:
First, the MFs are numbered starting with the leftmost MF (MF0). The even MFs will be stored in the "even" memory block, all odd MFs will be stored in the "odd" memory block. Since the overlap degree is 2, none of the MFs overlap each other in each memory block.[1] In order to reduce the amount of the required RAM, the MFs are stored in a compressed form (Fig. 2.b, 2.c): The first memory location k=0 contains the start value m_0. Each of the following memory locations $0 < k < k_{max}$ contains first the point on the x-axis x_k where the next change of the slope occurs and second the slope m_k. Obviously triangular and trapezoidal MFs can be stored in a very compact form. The storage of MFs with arbitrary shapes is also possible.



MF even

k	x	m
0.	—	m_0
1.	x_1	m_1
2.	x_2	m_2
3.	x_3	m_3
⋮		

MF odd

k	x'	m'
0.	—	m'_0
1.	x'_1	m'_1
2.	x'_2	m'_2
3.	x'_3	m'_3
⋮		

Fig. 2. Compressed storing of the MFs

[1] If the overlap degree q is greater than 2, we need q memory blocks.

For 8 membership functions (see Fig. 4) we need no more than 56 bytes of memory (8-bit resolution).
The reconstruction of the original MFs from the stored information can be done by the following procedure (Fig. 3.): At x=0, the start value m_0 will be read from address k=0 of the slope memory and stored into the adder. During the next system clock cycle (x=1), the contents of memory location k=1, m_1 (slope memory) and x_1 (position memory), will be read. During each of the subsequent system clock cycles, m_1 is added to the contents of the adder as long as $x \le x_1$. Now, the contents of the memory location k=2, m_2 and x_2, will be read. Again, as long as $x \le x_2$, m_2 is added at each system clock cycle to the contents of the adder. This procedure will be repeated for $x \le x_{max}$.

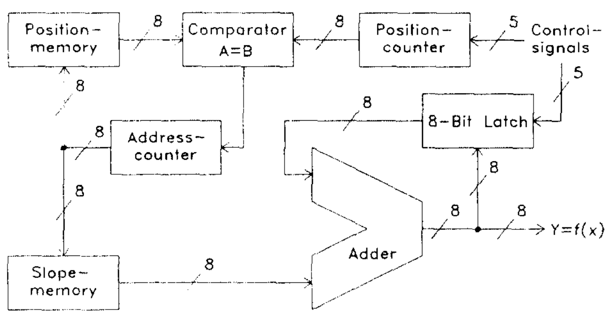


Fig. 3. MF-Generator

2. Fuzzification of PDM-signals

Fig. 4 depicts the architecture of the fuzzifier. In this and the following figures the latches which separate the pipeline stages are not included. With each system clock cycle new MF-values are produced by the MF-Generators. Therefore, a resolution of 8-bit results in 256 values for each main clock cycle. The MF-counter generates the numbers h and h+1 of the MFs which are actually the outputs of the MF-Generators. In order to save wires, only the number h is delivered. After the input signal I_n returns to zero, the corresponding MF-values and the MF-number h are stored for further processing. This completes the first stage of the pipeline.

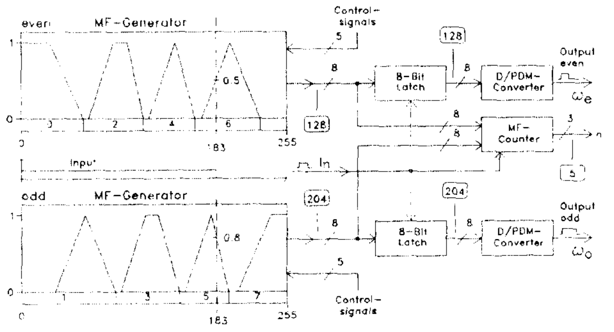


Fig. 4. Fuzzifier for PDM-Signals

The D/PDM converter transforms the stored MF-values into PDM-signals. This belongs to the second pipeline stage denoted prior by "rule base". An example is given in Fig. 4. The duration of the input signal I_n is 183. The output of the even MF-Generator delivers a MF-value of 128 since MF 6 is active. In the odd MF-Generator, MF 5 is active and the corresponding MF-value is 204. Therefore h=5 is stored in the MF-counter.

3. Rule base

We present two different architectures of a rule-base. The first one uses PDM-signals and is hardwired. So the hardware requirement is very small and there is a high flexibility. The second rule-base is programmable. This part of the controller is the second pipeline stage.

3.1. Hardwired rule-base with PDM-signals

In contrast to some traditional hardware realisations, this architecture allows the MIN- and MAX-operations on the fuzzified values without increasing the speed of operation and the amount of hardware substantially. It is also allowed to combine rules with the MIN- or MAX-operators. An example is given in Fig. 5.

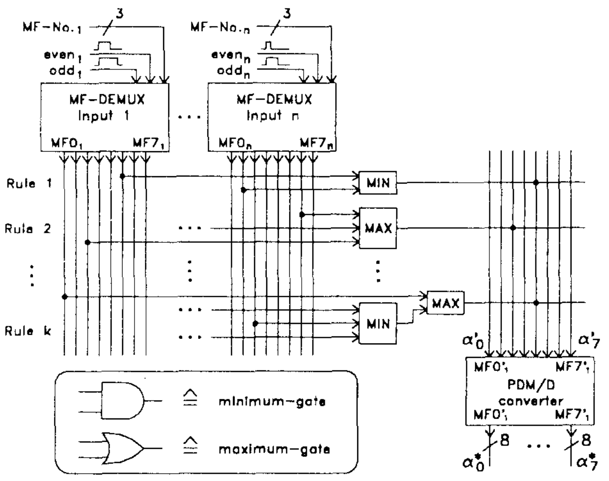


Fig. 5. Architecture of the rule-base using PDM-Signals

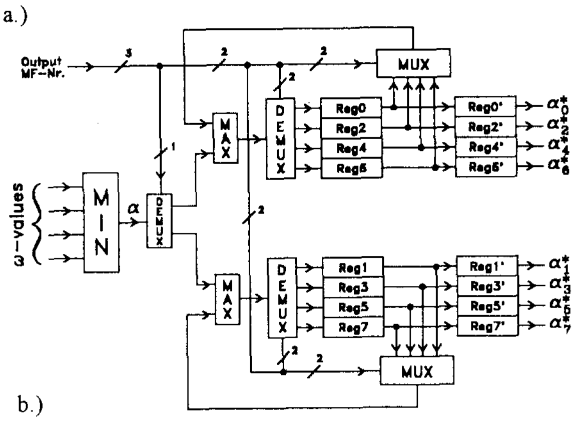
Depending on the value of the MF-number h, the MF-DEMUX-unit switches the even and the odd signal to the wires MF0 to MF7. If two or more rules using the same output MF' (e.g. see Fig. 8: rule 1 and k), the rule with the highest α-value will be used. This is easily realized using a wired OR. The MF'-MUX output signals α'_i are then converted by the PDM/D converter into digital values α^*_i and stored. This is the end of the second pipeline stage.
Obviously the rule base architecture offers flexibility, high speed operation and small chip area.

3.2. Programmable rule-base

Fig. 6.a. shows a typical rule-base and how to store it. The architecture of the unit for calculating the rules is shown in Fig. 6.b. . This unit works sequential and with every clock cycle one rule will be calculated. Using a resolution of 8 bit 256 rules could be calculated in one main clock cycle. For most of the applications this is enough.
The 'if-part' of the rules steers which ω_X-value of the MF-generators is switched to the MIN-gate. For example see Fig. 4. . If the input value is 183, the MFs 5 and 6 are active. Depending on the number of the subpremisse (stored in the rule-base memory) ω_o respectively ω_e is switched to the MIN-gate if this number is equal to h or h+1. Otherwise 'zero' is switched to the MIN-gate. This will be done in parallel for every input of the MIN-gate. The output value of the MIN-gate represents the trouth value of the rule. If this value is greater than the value stored in the corresponding register (on of Reg0 - Reg7 which represents the output MFs), the new value will be stored. After 256 clock cycles the values of the registers Reg0 - Reg7 will be transferred into Reg0'- Reg7' and could be used in the next pipeline stage.

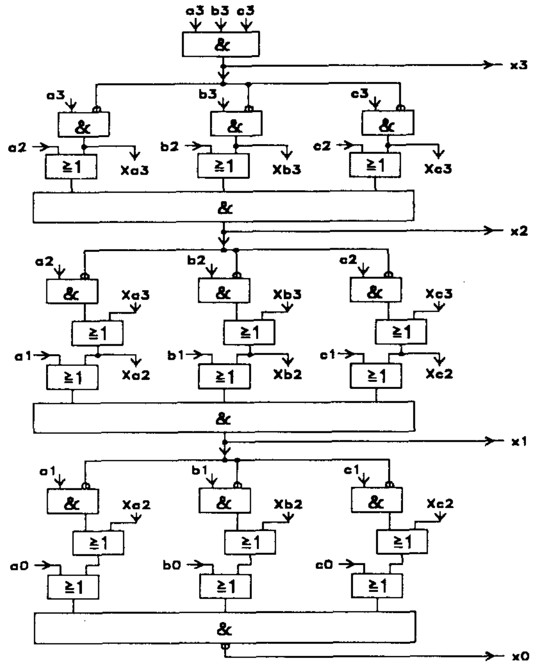
```
if In1 is A1 and In2 is B5 ... then O is X3 ⇒ 001 101 ... 011
if In1 is A4 and In2 is B2 ... then O is X5 ⇒ 100 010 ... 101
if In1 is A0 and In2 is B7 ... then O is X3 ⇒ 000 111 ... 011
                                   :
```

a.)

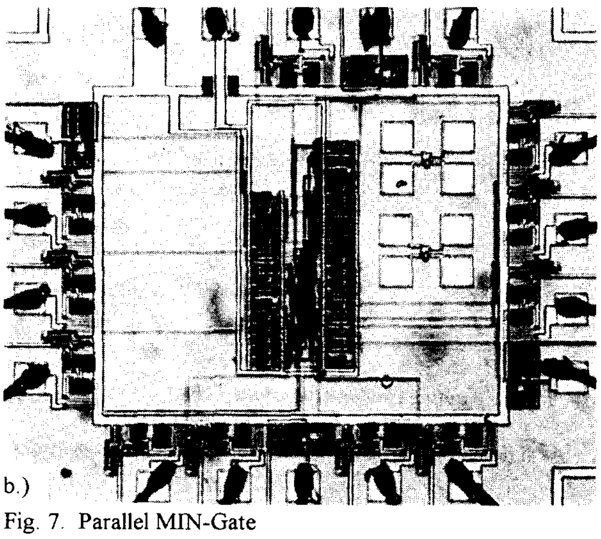


b.)

Fig. 6. Programmable rule-base



a.)



b.)

Fig. 7. Parallel MIN-Gate

Fig. 7. shows an interesting architecture of a multiple input MIN-gate e.g. used in our unit for calculating the rules. The operation speed of this gate is nearly independent on the number of inputs and it is very easy to expand the number of inputs. Additionally the hardware requirement is very low. A prototype of a 2.5μm CMOS implementation is shown in fig. 7.b. .

4. Inference unit

The new architecture of the inference unit described in this chapter is based on the assumption that the degree of overlap of the membership functions is 2. Again two memory blocks denoted by odd and even are used for storing the output MFs. Fig. 8 depicts the architecture for one controller output with inference unit (third pipeline stage) and defuzzification. The operation of the MF-Generators are basically the same as described before. The output values of the rule base α^*_i (see Fig. 6) are fed into two 4x8-bit multiplexers, steers by the number of the active output MF.
The inference for the even MFs will be shown in the following. On every system clock cycle the MF-Generator delivers a new value. Depending on the number of the active output MF the MUX switches the value α^*_i of this MF* to the MIN-gate. The MIN-gate limits the values of the MF-Generator to the output of the MUX. Obviously the output of this MIN-gate corresponds to the "even" part of the resulting output function. The output of the "odd"-part is generated in the same way. The MAX-gate combines the "even"- and the "odd"-part to one resulting output function.

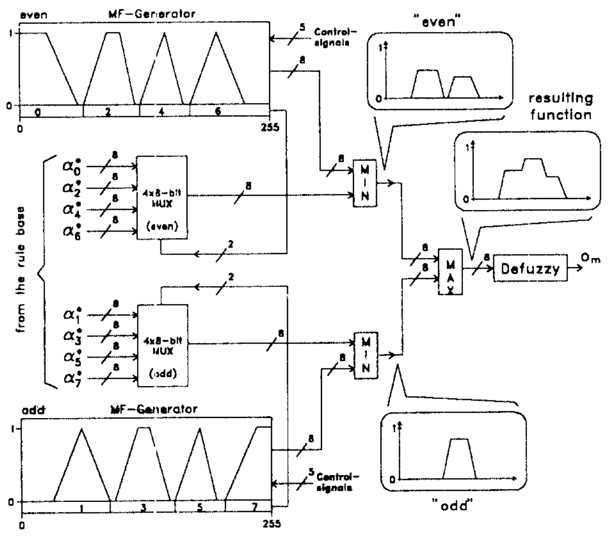


Fig. 8. Inference unit and Defuzzification

The advantages of the architecture are reduced memory (only two memory blocks) and a constant operation time not depending on the number of active rules.

5. Defuzzification

The controller uses the COG method for defuzzification. For calculating the center of gravity we have used the algorithm shown in /1/. For an optimal pipeline architecture it was necessary to integrate the adders for calculation of the counter and of the denominator in the third pipeline stage. The division is executed in the last pipeline stage. Because every stage of the pipeline takes one main clock cycle, we implemented the devider by an adder to save chip area.

6. Control unit

The control unit steers the pipeline and generates the following signals:

1. The system clock cycle (input clock dividing by four).
2. The main clock cycle (system clock cycle dividing by the x-resolution).
3. A power-up reset.

SUMMARY

The use of PDM-signals allows to realize an architecture of a fuzzy controller with reduced chip area. A prototype of the controller with two inputs, one output and a resolution of 8-bit is implemented on FPGAs. We programmed the controller for an inverted pendulum with 19 rules and 7 MFs for each input and for the output. This takes less than 850 CLBs of the used four FPGAs (Xilinx: XC3090-100). This corresponds to about 10.000 gates. With an input clock of 6 MHz we achieved 6.000 controller operations per second. Future work will speed up the controller with an ASIC-realisation. We expect to increase the operation time by at least 5 times.

ACKNOWLEDGEMENT

The authors would like to thank the VW-Stiftung Hannover for supporting this work and Anke Goos for their assistence for preparing this paper.

REFERENCES

/1/ H. Watanabe, W.D. Dettloff and K.E. Yount, "VLSI Fuzzy Logic Controller with Reconfigurable, Cascadable Architecture," IEEE Journal of Solid-State Circuits, vol. 25, no. 2, pp. 376-382, April 1990
/2/ H. Eichfeld, M. Löhner and M. Müller, "Architecture of a Fuzzy Logic Controller with optimized memory organisation and operator design," Int. Conf. on Fuzzy Systems, FUZZ-IEEE '92, San Diego, March 8-12, 1992, pp. 1317-1323
/3/ Ansgar P. Ungering, Bashar Qubbaj und Karl Goser, "Geschwindigkeits- und speicheroptimierte VLSI-Architektur für Fuzzy-Controller," VDE-Fachtagung: "Technische Anwendungen von Fuzzy-Systemen", Dortmund, 12./13. November 1992, S. 317-325
/4/ H.N. Teodorescu and T. Yamakawa, "Architectures for Rule-Chips Number Minimizing in Fuzzy Inference Systems," Proceedings of the 2nd International Conference on Fuzzy Logic & Neural Networks, Iizuka, Japan, July 17-22, 1992, pp. 547-550
/5/ Omron Corp, "Programmable fuzzy logical circuit - converts input signals into pulse signal, with programmable logical circuit outputting pulse," patent, PN J03122720 A 910524 DW9127, 1989
/6/ Ansgar P. Ungering, Karsten Thuener, Karl Goser, "Architecture of a PDM VLSI Fuzzy Logic Controller with Pipelining and Optimized Chip area" Second IEEE Int. Conf. on Fuzzy Systems, FUZZ-IEEE '93, San Francisco, March 28 - April 1, 1993, pp. 447 - 452
/7/ W.J.M. Kickert and E.H. Mamdani, "Analysis of a fuzzy logic controller", Fuzzy Sets and Systems 1, pp. 29-44, 1977
/8/ L.A. Zadeh "Fuzzy-Logik," IEEE Computer, pp. 83-92, April 1988
/9/ H. Surmann, B. Moeller and K. Goser, "A distributing self-organizing fuzzy rule-based system," Neuro Nimes 92, november 2-6, 1992, pp. 187-194
/10/ F. Deffontaines, A. Ungering, V. Tryba and K. Goser, "The concept of a RISC architecture for combining fuzzy logic and a Kohonen map on an integrated circuit," Neuro Nimes 92, november 2-6, 1992, pp. 555-564