# Embedded Fuzzy Controller for Industrial Applications

**Ferenc Farkas, Sándor Halász**

Department of Electric Power Engineering, Budapest University of Technology and Economics, ferenc.f.farkas@ericsson.com

*Abstract: The concept of the fuzzy logic makes feasible the creation of fuzzy controllers with low cost 16 bit microcontroller having the same performance as of controllers realized with more expensive Digital Signal Processor (DSP). In this article the implementation of such a fuzzy controller is proposed for 16 bit microcontroller with fast fuzzyfication-inference-defuzzyfication algorithm. Because the microcontroller receives information from the process via Analog-Digital Converter(s) and controls the process with the help of Digital-Analog Converter(s) the implemented algorithm does not use floating-point operations, only integer ones. However, for some type of fuzzy controllers, the error made by this algorithm is not greater than the error of a DSP based floating point algorithm.*

*Keywords: fuzzy logic, microcontroller, embedded systems*

# 1 Introduction

## 1.1 Fuzzy Controller and Embedded Systems

The world of embedded control is experiencing a push into the realm of fuzzy logic. Even household machines are advertised as being intelligent with the help of the built-in fuzzy logic. The popularity of the fuzzy logic is due to its simplicity and effectiveness in solving control problems. Conference proceedings and related periodicals contain myriads of articles presenting the advantages of control systems using fuzzy logic. Although fuzzy controllers are not able to solve every control problems, and have some disadvantages as well [7], one of the main disadvantage of using the fuzzy controller in embedded systems is the great number of floating point calculations made in real-time. This huge calculation capacity requires the use of Digital Signal Processor (DSP), which is more expensive compared to a 16 bit microcontroller (μC).

In the past, manufacturers have contended with the performance versus cost tradeoffs with no apparent fulfillment of both. Although, in the last decade the complexity of DSPs has evolved while their price decreased, manufacturers are always interested in cost reduction due to permanent competition. In this article a short comparison is presented between a DSP and a microcontroller based fuzzy controller, pointing out that – depending on the type of the fuzzy controller used and the precision of the Analog-Digital Converters (ADC) and Digital-Analog Converters (DAC) built in the system, – in most of the cases the DSP based fuzzy controller is not able to outperform the microcontroller based counterpart.

## 1.2 Microcontroller versus DSP

The main features of embedded systems are the compact realization, robustness, and cheapness. Cheapness can be achieved by using low performance microcontroller connected to low capacity memory. Naturally, such a system cannot be compared to a more complex DSP from the calculation capacity point of view. Thus, microcontrollers can be used in limited applications, where floating-point calculations are not required, or their use is limited. At first sight, microcontrollers are not suitable for realizing fuzzy controllers, due to hundreds of floating-point arithmetic done in real time. This short come might be overcome with a look-up table, storing the response of the fuzzy controller for different input/output combinations. However, the memory capacity is a strong limitation, so the table dimension is. Thus, only a limited number of input(s)/output(s) pairs are stored in the table, and interpolation is used in between. This solution has two major drawbacks: 1) the interpolation is not a good approximation for nonlinear functions, and the table dimension limits the number of useful pairs stored; 2) the fuzzy controller is rigid, cannot adjust its parameter to the changing environment as it is proposed in [1]-[2].

The concept of fuzzy logic makes feasible the use of a fuzzy controller built on low cost 8 or 16 bit microcontroller for some applications. Manufacturers, like MOTOROLA, have recognized the power of fuzzy logic and have created fuzzy kernels and support tools for a number of their 8 bit and 16 bit microcontrollers. However, these support tools lack the generalization and mathematical reasoning.

Although DSP is mostly applied in those applications where huge floating-point operations are performed in real time, its high price does not help the spreading of fuzzy controller in mass production. On the other hand a conventional microcontroller (like the INTEL's 80186 microcontroller) can be bought for a very low price. The latter does not support directly the floating-point operations, but does support the operations of integer type, like addition, subtraction, division and multiplication. For this reason, one must think about an algorithm, which incorporates only integer operands.

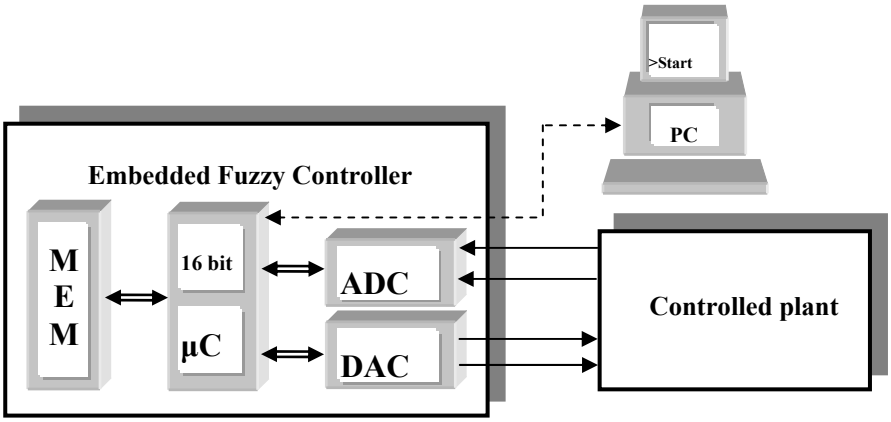


Figure 1
The concept of embedded fuzzy controller with ADC and DAC

In industrial applications, such as motion control, electrical drives, temperature and humidity stabilization, the fuzzy controller receives information from the controlled process via ADC(s) and controls it through DAC(s) as in Figure 1. For this reason, using a DSP in such cases is not far from the idea of making a fuzzy controller seeming better, faster and more accurate than it really is. Because the precision of ADC and DAC is always less than or equal to 16 bits, the use of 16 bit operands seems to be a reasonable compromise between accuracy and speed. Thus, the value stored in such an operand will be in the range of [0.65535]. Moreover, using appropriate fuzzy operations, the error made by the proposed algorithm is comparable to the DSP based fuzzy controller.

# 2 Theoretical Considerations

## 2.1 Starting from the Basic Idea

Let's consider two continuous and closed intervals $(a,b)$, $(E,F)$ on which the Euclidean distances are defined. For arbitrary $x \in (a,b)$, and $X \in (E,F)$ the following relation is held:

$$\frac{x-a}{b-a} = \frac{X-E}{F-E}. \tag{1}$$

Let's define the two intervals as being:

$$\begin{cases}(a,b) \equiv (0.0,1.0) \\ (E,F) \equiv (0,MAXINT), MAXINT = 65535\end{cases}, \tag{2}$$

that is, the $(a,b)$ interval represents the real numbers between $0.0$ and $1.0$, while the $(0,MAXINT)$ interval represents the integer numbers from $0$ to $MAXINT \equiv 2^{16}$. Because $(E,F)$ interval defined in this way is not continuous, there is no one-to-one mapping anymore. That is, $x' \in (a,b)$ being defined closed enough to $x \in (a,b)$ in terms of Euclidean distance, $x'$ will be mapped on the same $X \in (E,F)$ as $x$. The following notation will be used in the rest of this article: with lower case real numbers, while with capital letters integer numbers are denoted.

The following relations are obtained by rearranging relation (1), taking into account the definitions of the intervals, and replacing $x$ with $x'$, or $x$ with $y'$, $Y$ with $X$:

$$x' = \frac{X}{MAXINT}, \tag{3}$$

$$Y = [y' \cdot MAXINT], \tag{4}$$

where $[x]$ represents the integer part of $x$. These relations show how a real value can be mapped on an integer one, and vice versa.

It is by no surprise, that the defined $(a,b) \equiv (0.0,1.0)$ interval is the input/output of the fuzzy controller, while the $(E,F) \equiv (0,MAXINT)$ interval is the input/output of the 16 bit AD/DA converters. Mapping of real value to integer and vice versa is performed by the ADC and DAC, respectively. However, the fuzzy controller implemented on a DSP requires real number(s) for its input(s), and outputs real

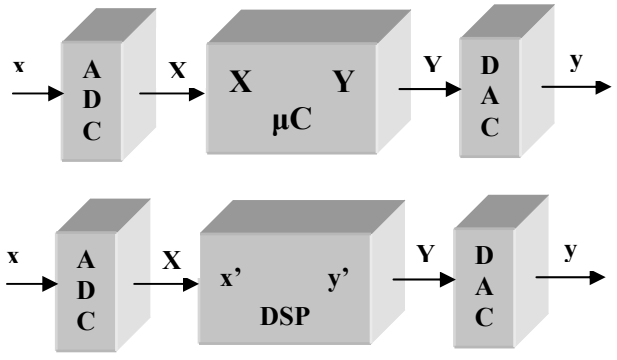


Figure 2
Microcontroller versus DSP. Which one is more suitable for an application?

number(s), as well. Thus, an algorithm for the DSP must convert the integer obtained from the ADC before applying to the fuzzy controller. Another algorithm should also convert the real number obtained from the output of the fuzzy controller to an integer one before feeding the DAC with the proper value. If it is so, there are two important questions: 1) what is the gain from using floating-point operands and operators? 2) can be implemented a fuzzy controller just using integer operands and operators?

In Figure 2 it is shown the two alternative fuzzy controllers: the upper one implemented on a microcontroller (μC), while the lower one on a DSP. The $x$ value obtained from the controlled plant is converted by the ADC to $X$, which is directly used by the fuzzy controller implemented on a microcontroller. However, the DSP needs to convert this $X$ value to another real value $x'$ using relation (3). It is obvious, that generally $x \neq x'$, that is, the input value is not equal to the input of the fuzzy controller. The fuzzy controller implemented on the DSP creates the output value $y'$, which is converted by another algorithm using relation (4) to $Y$ value. This $Y$ value (obtained directly from the microcontroller based fuzzy controller) is further converted to a real value $y$ by DAC, and this $y$ serves as a control signal. Again, it can be stated that generally $y' \neq y$, that is, the control signal is not equal to the output of the fuzzy controller. This gives a hint that floating-point calculation might be useless, because of the presence of AD and DA converters.

## 2.2 Error of the Integer Operators

Before diving into the deep water, it should be analyzed the behavior of the integer multiplication and division. Let's consider two arbitrary input values $X_1$, $X_2$ converted by the ADCs, and an output $Y$ supplied by the controller which is fed to the DAC. The output of the multiplication operation is

$$Y = [y' \cdot MAXINT] = [(x_1' \cdot x_2') \cdot MAXINT], \tag{5}$$

and using relation (3) the output is obtained in function of the inputs:

$$Y = \left[ \left( \frac{X_1}{MAXINT} \cdot \frac{X_2}{MAXINT} \right) \cdot MAXINT \right] = \left[ \frac{X_1 \cdot X_2}{MAXINT} \right]. \tag{6}$$

Similar result is obtained for the integer division, when the output is

$$Y = [y' \cdot MAXINT] = [(x_1' / x_2') \cdot MAXINT], \tag{7}$$

and using relation (4) the output in function of the inputs is obtained:

$$Y = \left[ \left( \frac{Y_1}{MAXINT} / \frac{Y_2}{MAXINT} \right) \cdot MAXINT \right] = \left[ \frac{Y_1 \cdot MAXINT}{Y_2} \right]. \tag{8}$$

It is important to note, that always the multiplication in the numerator must be performed firstly, and the obtained 32 operand should be divided by the 16 bit denominator (these operations are directly supported by the INTEL 80186 microcontroller using the MUL and DIV opcodes). The obtained result consists of the integer part of the division and the 16 bit remainder. Because only the integer part of the division must be fed to the DAC, there is no difference between using a DSP with floating-point operators, or a microcontroller with integer operators. Similar result is obtained when the weighted average is calculated (combination of multiplication and division):

$$Y = [y' \cdot MAXINT] = \left[ \frac{a \cdot x_1' + b \cdot x_2'}{a + b} \cdot MAXINT \right], \tag{9}$$

$$Y = \left[ \frac{a \cdot MAXINT \cdot x_1' + b \cdot MAXINT \cdot x_2'}{a \cdot MAXINT + b \cdot MAXINT} \cdot MAXINT \right], \tag{10}$$

$$Y = \left[ \frac{A \cdot x_1' \cdot MAXINT + B \cdot x_2' \cdot MAXINT}{A + B} \right] = \left[ \frac{A \cdot X_1 + B \cdot X_2}{A + B} \right]. \tag{11}$$

It can be seen, that only for one multiplication, division or a combination of them the error of the integer operations is not propagated through the DAC converter. In the next subchapter, an investigation for the whole fuzzy controller is performed.

## 2.3 Error Propagation through the Fuzzy Controller

### 2.3.1 Fuzzyfication

One of the most common used fuzzy controller is the Mamdani type controller, with MIN-MAX operators, and triangle or trapezoidal membership functions. Let's denote $\{a_x^i, b_x^i, c_x^i, d_x^i\}$ the parameters of the $i$ th input membership function, while $\{a_y^j, b_y^j, c_y^j, d_y^j\}$ the parameters of the $j$ th output membership function. In case of triangle membership function it can be simply considered $b_x^i = c_x^i$ or $b_y^j = c_y^j$. The simplest fuzzyfication is the singleton one, when the fuzzyfication function is the identity function, e.g. *f(y)=y*. That means that input variables are the singleton fuzzy inputs. The membership value $\mu^i(x)$ of the input $x$ corresponding to a membership function defined by the $\{a_x^i, b_x^i, c_x^i, d_x^i\}$ parameters is obtained by relation (12):

$$\mu^i(x) = \begin{cases} 0, x \le a_x^i \vee x \ge d_x^i \\ \dfrac{x - a_x^i}{b_x^i - a_x^i}, x < b_x^i \\ 1, b_x^i \le x \le d_x^i \\ \dfrac{d_x^i - x}{d_x^i - c_x^i}, c_x^i < x < d_x^i \end{cases}. \tag{12}$$

Let's consider an input $x$, where $a_x^i < x < b_x^i$, and calculate the $\mu^i(x)$ membership value for both floating-point and integer operators:

$$Y = \mu(X) = [\mu(x) \cdot MAXINT] = \left[ \frac{x - a_x^i}{b_x^i - a_x^i} \cdot MAXINT \right], \tag{13}$$

$$Y = \mu(X) = \left[ \frac{x \cdot MAXINT - a_x^i \cdot MAXINT}{b_x^i \cdot MAXINT - a_x^i \cdot MAXINT} \cdot MAXINT \right] = \left[ \frac{(X - A_x^i) \cdot MAXINT}{B_x^i - A_x^i} \right] \tag{14}$$

where $a_x^i, b_x^i \in [0.0, 1.0]$ and $A_x^i, B_x^i \in [0, MAXINT]$ represent the real, respective integer parameters of the $i$ th input membership function. In similar way should be calculated the $\mu^i(x)$ membership value for inputs with $c_x^i < x < d_x^i$. It was assumed in relation (14) that $a_x^i \cdot MAXINT = A_x^i$ and $b_x^i \cdot MAXINT = B_x^i$, which is not always true. In equation (3) the remainder of the division is not zero, that is, $a_x^i = \dfrac{A_x^i}{MAXINT} + a_{x\text{-}RES}^i$, where $a_{x\text{-}RES}^i$ is the remainder. In order to avoid any error, $a_x^i \cdot MAXINT = A_x^i$ must be held. This might look a restriction at first sight, but even in case of floating-point operands, the parameters of the membership functions are chosen to 2-3 places of decimals. Even with this restriction the parameters of the membership functions can be set for $MAXINT + 1 = 65536$ different value, which is enough for most of the applications. In conclusion, the fuzzyfication and the calculation of the membership value do not cause additional rounding error.

### 2.3.2 Inference

When the fuzzy controller has more then one input, generally the rule base is constructed in such way that AND relation exist between the rules. This AND rule is performed by the $MIN$ operator in case of Mamdani type controller. That means, the "firing" degree of the $k$ th rule is given by the following relation:

$$Y = \lambda(k) = [MIN\{\mu(x_1), \mu(x_2)\} \cdot MAXINT], \tag{15}$$

from where it can be concluded that there is no additional rounding error.

However, there are types of fuzzy controller where the AND relation is performed by multiplication. In those cases it is not so simple to decide what rounding errors one might have. Let's take two inputs, one belonging to the first, the other one belonging to the second membership function. Then the "firing" degree of the $k$ th rule is

$$Y = \lambda(k) = [\mu(x_1) \cdot \mu(x_2) \cdot MAXINT] = \left[ \frac{X_1 - A_x^1}{B_x^1 - A_x^1} \cdot \frac{X_2 - A_x^2}{B_x^2 - A_x^2} \cdot MAXINT \right], \tag{16}$$

where both the numerator and the denominator is multiplied by $MAXINT$ . There are two alternative solutions. An algorithm should be implemented in the first case which is able to divide a 48 bit number by a 32 bit number:

$$Y = \lambda(k) = \left[ \frac{(X_1 - A_x^1) \cdot (X_2 - A_x^2) \cdot MAXINT}{(B_x^1 - A_x^1) \cdot (B_x^2 - A_x^2)} \right]. \tag{17}$$

Although, this kind of algorithm has no additional rounding error – the 48 bit is divided by the 32 bit number, but only the integer part must be considered –, the running time of the algorithm might be significant for some applications. Another solution is also presented, which runs faster, but has rounding error. Let's multiply in equation (16) both the numerator and denominator with $MAXINT$ :

$$Y = \lambda(k) = \left[ \frac{\left( \frac{(X_1 - A_x^1) \cdot MAXINT}{B_x^1 - A_x^1} \right) \cdot \left( \frac{(X_2 - A_x^2) \cdot MAXINT}{B_x^2 - A_x^2} \right)}{MAXINT} \right] = \left[ \frac{\mu(X_1) \cdot \mu(X_2)}{MAXINT} \right] \tag{18}$$

That means, membership values are calculated individually using relation (12), then the obtained membership values are multiplied together and the resulted product is divided by $MAXINT$ . Although, this simplified algorithm is supported by the MUL and DIV opcodes of the microcontroller, rounding error is introduced. This is due to the fact, that the two members in the numerator are not calculated precisely, only the integer part is taken into account (this is equivalent with one replacing the round brackets with brackets in the numerator). The rounding error can be estimated if one considers $P_1, P_2, Q_1, Q_2$ arbitrary integer numbers, and calculates the product of their quotient:

$$\begin{cases} \frac{Q_1}{P_1} \cdot \frac{Q_2}{P_2} = \frac{N_1 P_1 + R_1}{P_1} \cdot \frac{N_2 P_2 + R_2}{P_2} = \left( N_1 + \frac{R_1}{P_1} \right) \left( N_2 + \frac{R_2}{P_2} \right) \\ \frac{Q_1}{P_1} \cdot \frac{Q_2}{P_2} = N_1 N_2 + N_1 \frac{R_2}{P_2} + N_2 \frac{R_1}{P_1} + \frac{R_1}{P_1} \cdot \frac{R_2}{P_2} \end{cases}, \tag{19}$$

where $N_1, N_2$ are the integer parts of the quotients and $R_1, R_2$ are the remainders.

It can be concluded from relation (19) that only the $N_1 \cdot N_2$ product is considered in equation (18), the rest three terms are omitted. The following inequality is held for the last three terms of relation (19):

$$N_1 \cdot \frac{R_2}{P_2} + N_2 \cdot \frac{R_1}{P_1} + \frac{R_1}{P_1} \cdot \frac{R_2}{P_2} < (MAXINT - 1) + (MAXINT - 1) + 1 = 2 \cdot MAXINT - 1 \,, \quad (20)$$

because both $R_1, R_2$ are less than $MAXINT$. From relation (18) it follows that additional rounding error occurs only if the following condition is held:

$$R_\mu + N_1 \cdot \frac{R_2}{P_2} + N_2 \cdot \frac{R_1}{P_1} + \frac{R_1}{P_1} \cdot \frac{R_2}{P_2} \geq MAXINT \,, \quad (21)$$

where $R_\mu = \mu(X_1) \cdot \mu(X_2) - N_\mu \cdot MAXINT$ is the remainder of division of the equation (18). From equation (18) and inequality (20) it can be concluded that the rounding error is between 0 and 2 bits. 0 bit error occurs when inequality (21) is not held, that is, when $R_1, R_2$ remainders are far less than $P_1, P_2$ and $R_\mu$ is also minor. 2 bit error occurs only and only if equality $R_\mu + N_1 \cdot \frac{R_2}{P_2} + N_2 \cdot \frac{R_1}{P_1} + \frac{R_1}{P_1} \cdot \frac{R_2}{P_2} = 2 \cdot MAXINT$ holds. However, this is a rear situation. Thus, it can be concluded that the average additional rounding error is 1 bit. Moreover, this 1 bit rounding error can even be absorbed by the imprecision of the DAC, taking into account that common DACs have only ±½ bit precision at conversion. Important to note, that since the 2 least significant bits are not passed to the 14 bit DAC, even the 2 bit error is not present at the output of the DAC.

$MIN$ operator is used for the inference operator, as well, although there are fuzzy controllers where the multiplication is used instead. In case of $MIN$ operator the height of the $j$ th output membership function is defined by the minimum of the "firing" degree of the rules containing the same antecedents:

$$Y = h_y^j = [MIN\{\lambda(i), \lambda(k)\} \cdot MAXINT] \,, \quad (22)$$

which does not introduce additional rounding error. However, this is not true when multiplication is used instead of $MIN$ operator. In case of multiplication the height of the $j$ th output membership function is defined by the product of the "firing" degree of the rules containing the same antecedents:

$$Y = h_y^j = [\lambda(i) \cdot \lambda(k) \cdot MAXINT] \,. \quad (23)$$

There two possibilities: the $\lambda(i), \lambda(k)$ "firing" degrees either have been calculated with $MIN$ operator using equation (15) or with multiplication operator as in case of relation (16). In the former case

$$Y = h_y^j = [MIN\{\mu_i(x_1), \mu_i(x_2)\} \cdot MIN\{\mu_k(x_1), \mu_k(x_2)\} \cdot MAXINT] \,, \quad (24)$$

from where, without loosing the generality, it is considered that $\mu_i(x_1) < \mu_i(x_2)$ and $\mu_k(x_1) > \mu_k(x_2)$, from where it is obtained:

$$Y = h_y^j = [\mu_i(x_1) \cdot \mu_k(x_2) \cdot MAXINT]. \tag{25}$$

It is obvious, that equation (25) looks like equation (16), and thus, the same conclusion can be drawn: the additional rounding error is between 0 and 2 bits, and the average rounding error is 1 bit.

The situation is much complicated when the overall "firing" degree is calculated by multiplication. In that case the height of the $j$ th output membership function is

$$Y = h_y^j = [(\mu_i(x_1) \cdot \mu_i(x_2)) \cdot (\mu_k(x_1) \cdot \mu_k(x_2)) \cdot MAXINT], \tag{26}$$

from where it is obtained by substituting the membership values

$$Y = h_y^j = \left[ \frac{X_1 - A_x^i}{B_x^i - A_x^i} \cdot \frac{X_2 - A_x^i}{B_x^i - A_x^i} \cdot \frac{X_1 - A_x^k}{B_x^k - A_x^k} \cdot \frac{X_2 - A_x^k}{B_x^k - A_x^k} \cdot MAXINT \right]. \tag{27}$$

One might construct an algorithm which is able to calculate the 80 bit product of the numerator, which is divided by the 64 bit product of the denominator, in which case there is no rounding error. However, such algorithms will slower the inference of the fuzzy controller, which might not be suitable for some applications. For this reason, the following solution is proposed:

$$Y = h_y^j = \left[ \frac{\lambda(i) \cdot \lambda(k)}{MAXINT} \right], \tag{28}$$

where $\lambda(i), \lambda(k)$ "firing" degrees are calculated using the equation (18). In this case additional rounding error exists. In order to find out the magnitude of the rounding error, let's consider the worst case when both "firing" degrees have been calculated with 2 bit error.

That means, the $\lambda(i) \cdot \lambda(k)$ product should be replaced by $(\lambda(i)+2) \cdot (\lambda(k)+2) = \lambda(i) \cdot \lambda(k) + 2\lambda(i) + 2\lambda(k) + 4$ in order to calculate the precise value. This gives an additional rounding error if $R_\lambda + 2\lambda(i) + 2\lambda(k) + 4 \geq MAXINT$ inequality holds, where $R_\lambda = \lambda(i) \cdot \lambda(k) - N_\lambda \cdot MAXINT$ represents the remainder of the division from relation (28). The largest rounding error occurs when both "firing" degree is equal to $MAXINT$, and $R_\lambda \geq MAXINT - 4$ inequality holds. Although this rounding error of 5 bits seems very large and might not be acceptable, in common applications never occur. However, 4 bit errors might still persist, and the average rounding error is around 2 bits, taking into account that the $\lambda(i), \lambda(k)$ "firing" degrees are calculated with 1 bit rounding error in average. This 2 bit error might still bother the designer of a fuzzy controller. However, 14 bit DACs are used in a

large number of applications, in which case even 4 bit rounding error does not appear at the output of the 14 bit DAC, since the 2 least significant bits are not passed to the 14 bit DAC.

The aggregation of the output membership functions can be done in several ways [4]-[6]. The two most popular aggregations are the MAX operator and the bounded sum. When *MAX* operator is used for aggregating the output membership function

$$Y_{fuzzy} = \left[ \underset{j}{MAX} \{h_y^j\} \right] \tag{29}$$

does not contain additional rounding error. So it is, when aggregation is calculated with the bounded sum operator:

$$Y_{fuzzy} = \left[ \left\langle \sum_j h_y^j \right\rangle_{\leq MAXINT} \right]. \tag{30}$$

Thus, it can be concluded that the inference does not introduce additional rounding error if *MIN* operator is used. When product is used for the inference, the additional rounding error introduced might be slightly significant only if 16 bit DAC is used.

### 2.3.3 Defuzzyfication

Several defuzzyfication methods exists, some of them are more spread than the others [4]-[6]. One of the most popular defuzzyfication method is the Mean of Maximum (MOM), when the crisp output value is calculated as the mean of maximum values of the aggregated output fuzzy set:

$$Y = \left[ \sum_{k=1}^{M} \frac{h_y^k}{M} \right], \tag{31}$$

where $h_y^k$ denotes the heights of those points where the fuzzy set has one of its maximum value. As it can be seen, there is no additional rounding error introduced in this way. Thus, it can be concluded, that Mamdani type fuzzy controller with MIN-MAX operators and MOM defuzzyfication gives the same output as a DSP based fuzzy controller, even when 16 bit DAC is used.

Other well-known defuzzyfication methods are the Center of Area (COA) and Center of Gravity (COG) methods. The only difference between these two methods is that COA calculates the center of the aggregated fuzzy set, while COG calculates the center of the gravity of the fuzzy sets taking part in the aggregation. Thus, COG calculates twice the overlapped areas. In what follows, only the COG is presented, COA has similar reasoning. The COG defuzzyfication method is

$$Y = \left[ \frac{\sum_k T^k \cdot X_G^k}{\sum_k T^k} \right] = \left[ \frac{\sum_k \lambda(k) \cdot (D_x^i - A_x^i + C_x^i - B_x^i) \cdot X_G^k}{\sum_k \lambda(k) \cdot (D_x^i - A_x^i + C_x^i - B_x^i)} \right] \tag{32}$$

where $X_G^k$ represents the center of gravity of the $k$ th modified output membership function, while $T^k = \lambda(k) \cdot (D_x^i - A_x^i + C_x^i - B_x^i)$ is the area of this membership function. The use of an algorithm which calculates the 48 bit numerator and divides it with the 32 bit denominator is recommended in order to avoid significant rounding error. Analyzing only the division, it can be concluded that there is no additional rounding error, because anyway only the integer part is passed to the DAC. However, in the nominator the $\lambda(k)$ values are not the real ones, only the integer parts. This leads to an additional rounding error. In order to try to estimate the error introduced by the COG defuzzyfication method, let's consider $P_1, P_2, Q_1, Q_2$ arbitrary integer numbers, and calculate the weighted average of their quotient:

$$\frac{X_1 \frac{Q_1}{P_1} + X_2 \frac{Q_2}{P_2}}{\frac{Q_1}{P_1} + \frac{Q_2}{P_2}} = \frac{(N_1 X_1 + N_2 X_2) + X_1 \frac{R_1}{P_1} + X_2 \frac{R_2}{P_2}}{(N_1 + N_2) + \frac{R_1}{P_1} + \frac{R_2}{P_2}}. \tag{33}$$

It can be observed, that in relation (32) only the members of the round brackets from (33) is taken into account which leads to additional rounding error. However, it is important to notice, that omitting the terms in the numerator will cause a negative rounding error, while omitting the terms the denominator will cause a positive rounding error. Thus, their counter effect will partly extinguish the rounding error when the terms from both numerator and denominator are omitted. Because there is no simple way to analytically determine the additional rounding error caused by the COG calculated with (32), this error was determined experimentally (for a detailed result see subchapter 4.2). Experimental results show that the additional rounding error caused by COG is less than 4 bits. Here again, it can be concluded that using only 14 bit DAC the error introduced by the COG is extinguished by the DAC.

### 2.3.4 Sugeno Type Fuzzy Controller

Because the defuzzyfication has a significant calculation demand, another type of fuzzy controller is also used, the so called Sugeno type fuzzy controller. In this case the output of the rule is a polynomial function of the inputs, instead of a fuzzy set. The output of the controller is the weighted average of the output of the rules, where the weight is equal to the "firing" degree of the given rule. For simplicity, the most common used Sugeno type controller is the zero order one, in

which case the output of the rule is a crisp value. Using MIN-MAX operators, the output of the zero order Sugeno type fuzzy controller is given as

$$Y = [y' \cdot MAXINT] = \left[ \frac{\mu(x_1) \cdot x_1 + \mu(x_2) \cdot x_2}{\mu(x_1) + \mu(x_2)} \cdot MAXINT \right], \tag{34}$$

where – for simplicity – it was considered that there are only two inputs with two rules and the "firing" degree of the first rule is equal to the membership value of the first input, while the "firing" degree of the second one is equal to the membership value of the second input. The output of the first rule is equal to the first input, while the output of the second one is equal to the second input. Considering relations (9)-(11), it can be calculated the output of the zero order Sugeno type fuzzy controller as

$$Y = \left[ \frac{\frac{X_1 - A_x^i}{B_x^i - A_x^i} \cdot x_1 + \frac{X_2 - A_x^j}{B_x^j - A_x^j} \cdot x_2}{\frac{X_1 - A_x^i}{B_x^i - A_x^i} + \frac{X_2 - A_x^j}{B_x^j - A_x^j}} \cdot MAXINT \right]. \tag{35}$$

After rearranging it

$$Y = \left[ \frac{(X_1 - A_x^i) \cdot (B_x^j - A_x^j) \cdot X_1 + (X_2 - A_x^j) \cdot (B_x^i - A_x^i) \cdot X_2}{(X_1 - A_x^i) \cdot (B_x^j - A_x^j) + (X_2 - A_x^j) \cdot (B_x^i - A_x^i)} \right]. \tag{36}$$

The rounding error is zero when an algorithm is used which calculates the 48 bit numerator and divides it with the 32 bit denominator. A simplified and faster way is to use the MUL and DIV opcodes of the microcontroller. In order to reduce the error, both the numerator and denominator should be multiplied by $MAXINT$.

$$Y = \left[ \frac{\left( \frac{(X_1 - A_x^i) \cdot MAXINT}{B_x^i - A_x^i} \cdot X_1 + \frac{(X_2 - A_x^j) \cdot MAXINT}{B_x^j - A_x^j} \cdot X_2 \right)}{\frac{(X_1 - A_x^i) \cdot MAXINT}{B_x^i - A_x^i} + \frac{(X_2 - A_x^j) \cdot MAXINT}{B_x^j - A_x^j}} \right] = \left[ \frac{\mu(X_1) \cdot X_1 + \mu(X_2) \cdot X_2}{\mu(X_1) + \mu(X_2)} \right] \tag{37}$$

In conclusion, the zero order Sugeno type fuzzy controller calculated in this simple way (37) has a similar rounding error as the COG defuzzyfication method, the only difference is, that in this case a 32 bit numerator is divided by 16 bit denominator. Using only 14 bit DAC, this rounding error does not appear at the output of the DAC.

An important remark is that the presented error propagation of the fuzzy controller is true for 8 bit microcontroller as well, when 8 bit ADC and DACs are used. It can be also concluded that Mamdami type fuzzy controller with MIN-MAX operator and MOM defuzzyfication has no rounding error compared to the DSP

based one. This is also true for multiplication operator and COG defuzzyfication if one uses only 14 bit DAC.

# 3 Implementation of the Embedded Fuzzy Controller

In this chapter an embedded fuzzy controller is presented which has been implemented in an industrial computer equipped with a 16 bit microcontroller (INTEL 80186 @ 16 MHz).

## 3.1 Storing the Parameters of the Membership Functions

Cheapness of an embedded system can be achieved by using low performance microcontroller connected to low capacity memory. As it was pointed out in subchapter 2.1, the parameters of the membership functions are stored as 16 bit integers and so the variables, like inputs/outputs of the controller. Thus, the memory requirement for storing the parameters and variables of the algorithm are only half or even less than a quarter of the memory capacity needed for a DSP based controller. This is due to the fact, that floating point values are usually stored in 32 bit ("single" float) or 64 bit ("double" float), sometimes even 80 bit ("extended" float) memory storage.

In the proposed fuzzy controller 3 type of membership functions can be used for the input (trapezoidal, triangle, and the generalized bell curve) and 3 type for the output (trapezoidal, triangle, and singleton). Four parameters need to be stored for the trapezoidal membership function, let's denote them with $A$, $B$, $C$, and $D$. In the same way can be stored the parameters of the triangle membership function ($B = C$). The generalized bell curve

$$f(X) = \frac{1}{1 + \left| \frac{X - B}{C} \right|^{2D}} \tag{38}$$

needs 3 parameters to be stored, where $D$ – for the simplicity of the algorithm – only 4 values can take {0.5; 1; 1.5; 2} coded on 4 different integer values. Finally, only one value needs to be stored in $B$ for the singleton.

For a general solution every membership function has 4 parameters, as it is shown in Table 1.

Table 1
Parameters of the membership functions

| 1ST PARAMETER | 2ND PARAMETER | 3RD PARAMETER | 4TH PARAMETER |
|---|---|---|---|
| *A* | *B* | *C* | *D* |

Starting from the assumption that the 1st parameter of the trapezoidal (triangle) membership function does not reach the *MAXINT* value, just analyzing the value of the 1st parameter , it can be decided the type of the membership function stored in a memory location (If the 1st parameter is equal to *MAXINT* a singleton is defined, and can be identified with only one parameter). If parameter $A$ is less than *MAXINT*, a trapezoidal (triangle) membership function, otherwise a generalized bell curve for the input, or singleton for the output is stored. The number of the input and output membership function can be arbitrary large, however, for the proposed fuzzy controller it has been limited to 4, respective to 2. In the same way, the number of membership functions for an input/output can be arbitrary large, but in the proposed fuzzy controller it has been limited to 8. Although, the number of membership functions for an input/output is generally even number, 8 is a power of two, which helps omitting multiplication when accessing the memory location. Thus, the memory content, which stores the $k$ th parameter of $j$ th membership function of $i$ th input/output, is addressed in the following way (C programming style notation):

$$T[i][j][k] = T_{addr} + i << 6 + j << 3 + k << 2 + H / L, \tag{39}$$

where $<<$ denotes the left shift operator (which is equivalent with multiplying by the power of 2), and $H / L$ is set to zero for the lower byte, respective to one for the higher byte of the integer value. Important remark is, that all indices start from zero! $T_{addr}$ represents the absolute value of the first byte of the table, the rest represents the offset of a parameter. The memory is organized in byte form (the width of the data bus is one byte), and the memory capacity needed to store al the parameters is $8 \cdot 8 \cdot 6 = 384$ bytes. Additional 6 bytes are needed to store the number of the membership functions used for each input/output (if this value is set to zero, the given input/output is not used!).

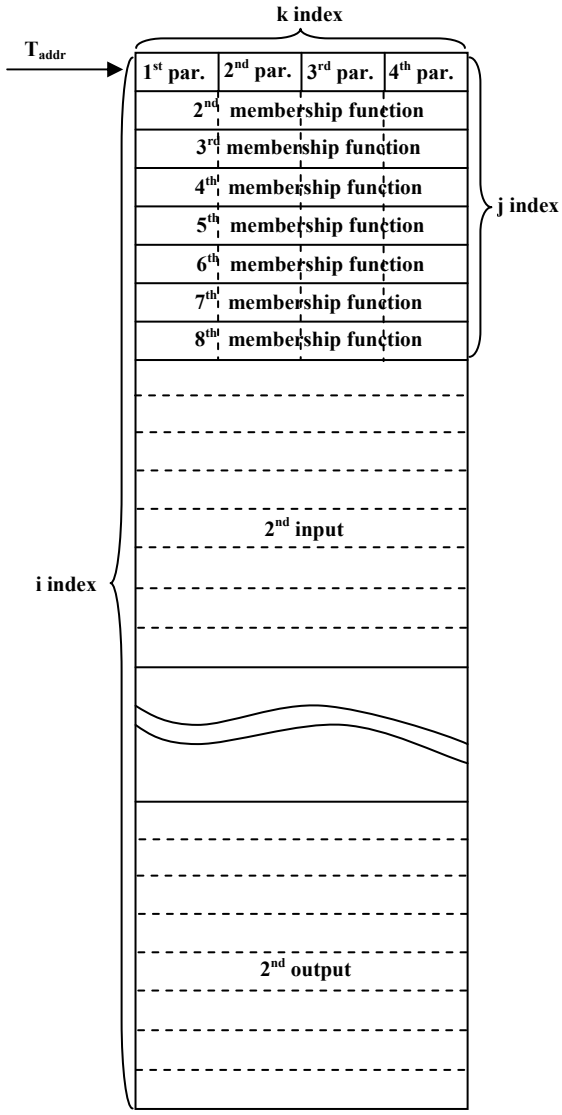


Figure 3
The parameters of the membership functions stored in memory

## 3.2 Constructing the Algorithm

In this chapter some useful hints are presented to construct the algorithm. See [3] for a simplified pseudo code of the implemented algorithm.

### 3.2.1 Fuzzyfication

For each input/output 2 bytes of memory location are used where the current input/output value of the fuzzy controller can be stored. This needs additional 12 bytes of memory location. The fuzzyfication function used in the implementation is the identity function, e.g. *f(y)=y*. That means, that input variables are not fuzzyfied, instead they are employed in the inference process directly. Although, the fuzzyfication process means only the fuzzyfication of each input variable, in what follows the calculation of the degree of consistency between the input value and the membership functions of the appropriate input is also included.

In case of trapezoidal (triangle) membership function the degree of consistency is calculated as in (14), which needs one multiplication and one division. In order to omit the multiplication, the following trick is used:

$$MAXINT \cdot X = FFFF_H \cdot X = (10000_H - 1) \cdot X = (X << 16) - X \,, \tag{40}$$

thus, subtraction is used instead of multiplication. In the worst case only a division is needed, otherwise the degree of consistency is either 0 or 1. In case of generalized bell curve the precision of the division can be increased if both the numerator and denominator are multiplied by 256 (shifted with one byte to the left). For example, when $D = 0.5$ the following relation gives an approximate value:

$$\mu(X) = \frac{FFFF00_H}{100_H + \dfrac{|X - B| << 8}{C}} \,. \tag{41}$$

Storing the $\mu(X)$ membership value needs two bytes, and current membership values are stored in similar way as the parameters of the membership functions.

### 3.2.2 Inference

Compactness and effectiveness has taken with first priority when the codification of the rules has been implemented. Each rule consists of two parts: one antecedent part containing one or more antecedent term, and one consequent part having one consequent term. Each antecedent/consequent term needs one byte for storage as it can be seen in Figure 4. The end of the rule-base is indicated by a value greater than 128 (the most significant bit is set to 1).

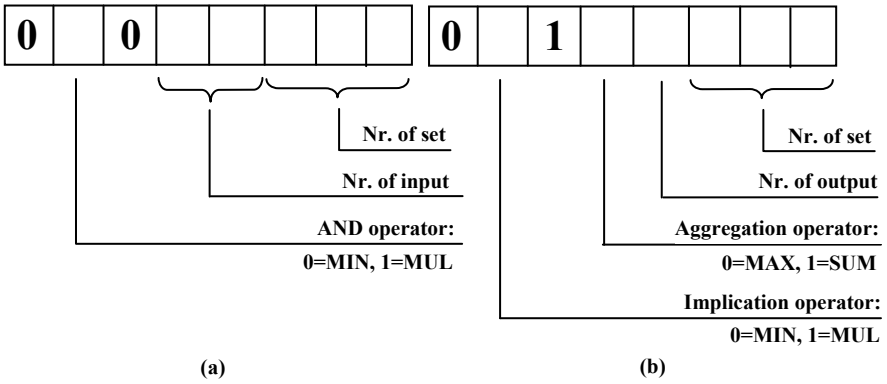


Figure 4
Coding the antecedent (a) and consequent (b) term of the rule

It should be noted that in the antecedent term the number of the membership function (set) and the number of input is stored in such way, that the memory location of the parameter can be calculated by masking the antecedent term, then shifting to the left by 3 and adding the number of the parameter, as it is in (39). In the same manner is coded the consequent term. In the antecedent term the AND operator (MIN - minimum or MUL - multiplication) is also coded. In the consequent term the implication operator (MIN - minimum or MUL - multiplication) and the aggregation operator (MAX - maximum or SUM - bounded sum) are coded. The number of bytes required for storing a single rule is maximum 5, since to each input one antecedent term corresponds, while the rule base requires in the worst case 20480 bytes (counted for 4096 rules). However, in everyday applications the number of rules is limited to around 100.

### 3.2.3 Defuzzyfication

When singletons are defined at the output (Sugeno type fuzzy controller) the defuzzyfication is simple and there is no need to describe in details. In case of COG it has been used the trick, that the division can be avoided if the divider is equal to *MAXINT* . This is due to the fact, that having a 32 bit dividend and 16 bit divider, the latter equal to *MAXINT* , the result is the high word of the dividend.

# 4 Results

## 4.1 Simulation Results

Most fuzzy engines are analyzed for three basic parameters: performance, code size, and inference time. Performance involves the smoothness of output in transition areas (i.e. where the input membership functions overlap). Performance was examined by building two fuzzy controllers in a MATLAB environment, one using floating point calculation, and the other one using only integer values. Both fuzzy controllers have 2 inputs and one output, having the following rule base:

**1. (In1==NB) & (In2==NB) => (Out=NB)**
**2. (In1==NB) & (In2==ZR) => (Out=NB)**
**3. (In1==NB) & (In2==PB) => (Out=ZR)**
**4. (In1==ZR) & (In2==NB) => (Out=NB)**
**5. (In1==ZR) & (In2==ZR) => (Out=ZR)**
**6. (In1==ZR) & (In2==PB) => (Out=PB)**
**7. (In1==PB) & (In2==NB) => (Out=ZR)**
**8. (In1==PB) & (In2==ZR) => (Out=PB)**
**9. (In1==PB) & (In2==PB) => (Out=PB)**

The obtained surfaces can be seen in Figure 5-8. In Figure 5 Mamdani type fuzzy controller is presented with MIN-MAX operator, trapezoidal membership function, and COG defuzzyfication. There is no visible difference between the DSP and μC based fuzzy controller, the average difference is only 1-2 bit when using 16 bit DAC. In Figure 6 Mamdani type fuzzy controller is presented with MUL-SUM operators, trapezoidal membership function, and COG defuzzyfication. It can be observed that the surface of the μC based fuzzy controller around zero is flatter than the surface of the DSP based controller when 16 bit DAC is used. This difference is due to the presence of the rounding error of the μC based fuzzy controller.

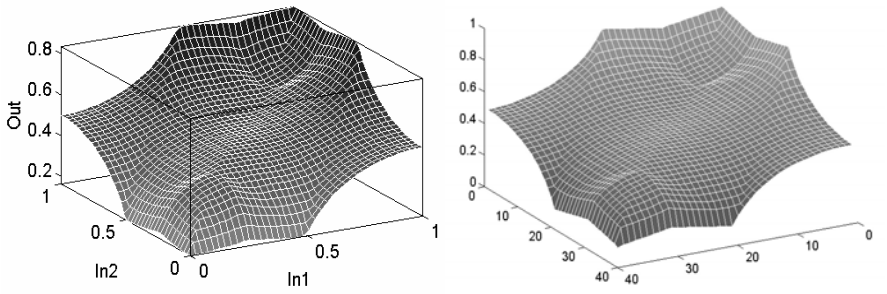


Figure 5
Surface of the Mamdani type fuzzy controller with MIN-MAX operators, trapezoidal membership functions, and COG defuzzyfication (DSP based on the left, μC based on the right)

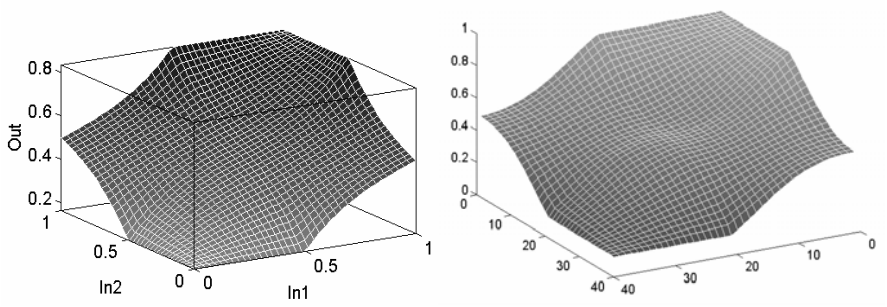


Figure 6
Surface of the Mamdani type fuzzy controller with MUL-SUM operators, trapezoidal membership functions, and COG defuzzyfication (DSP based on the left, μC based on the right)

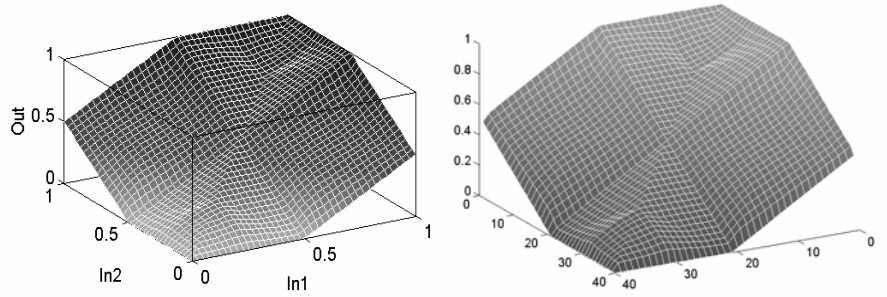


Figure 7
Surface of the Sugeno type fuzzy controller with MIN-MAX operators, and trapezoidal membership functions (DSP based on the left, μC based on the right)

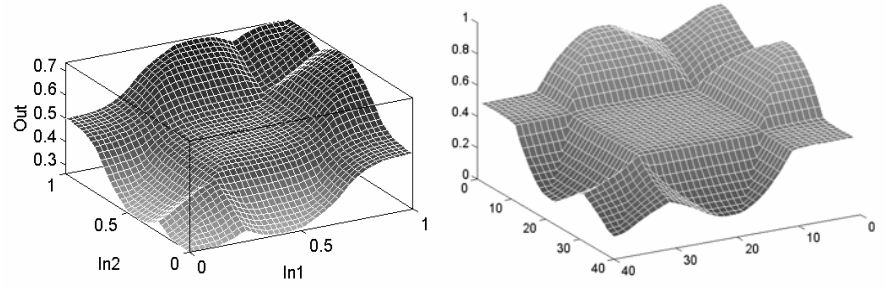


Figure 8
Surface of the Mamdani type fuzzy controller with MIN-MAX operators, generalized bell curve membership functions, and COG defuzzyfication (DSP based on the left, μC based on the right).

In Figure 7 the surfaces of the zero order Sugeno type fuzzy controller with MIN-MAX operators, and trapezoidal membership functions are presented. As it can be seen there is no visible difference between the DSP and μC based fuzzy controller, the average difference is only 1-2 bit when using 16 bit DAC. Finally, in Figure 8 the Mamdani type fuzzy controller with MIN-MAX operators, generalized bell curve membership functions and COG defuzzyfication is shown. In this case, the rounding error is significant, and is presented only for the sake of completeness, its usefulness might be questionable for some applications.

## 4.2 Experimental Results

The proposed embedded fuzzy controller was implemented in an industrial computer (UNI-PLC-100) at Process Control Ltd. This industrial computer is based on INTEL 80186 microcontroller at 16 MHz, and is equipped with analogue and digital input/output boards. The front panel of the industrial computer is presented in Figure 9.

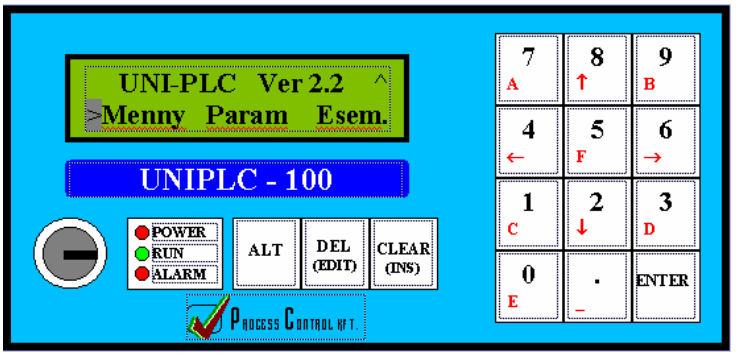


Figure 9
The front panel of the industrial computer. Courtesy of the Process Control Ltd.

There are two possibilities to store the parameters of the membership functions and the rule base: either entering manually with the help of the keyboard on the front panel, or downloading through the serial line from a Personal Computer (PC).

Because the whole fuzzy controller algorithm is implemented in assembler language, the code size is very compact. The overall code is less than 10 KB. The inference time is also impressive, taking into account that the microcontroller runs only at 16 MHz. Some inference time for Mamdani and Sugeno type fuzzy controller with MIN-MAX operators, trapezoidal membership functions and COG defuzzyfication (only for the Mamdani) are presented in Table 2. It can be seen that the total inference time (fuzzyfication+inference+defuzzyfication) is very impressive for a fuzzy controller with 2 inputs, 3 membership functions (sets) per input, 9 rules and COG defuzzyfication. It is only 1011 μs, which means that the fuzzy controller is able almost every 1 ms to update the control signal.

Even though a DSP based fuzzy controller might be able to update the control signal 100 times more frequently than the microcontroller based one, this is useless if the controlled process is slow, and its state variables changes so slowly, that there is no effect on the controlled process if one update the control signal more frequently.

Table 2
Inference time with MIN-MAX operators

| Nr. of input | Nr. of set/input | Nr. of rules | Fuzzification (trapezoidal) [μs] | Inference [μs] | Defuzzyfication Sugeno / COG [μs] |
|---|---|---|---|---|---|
| 1 | 3 | 3 | 78 | 50 | 85 / 641 |
| 1 | 5 | 5 | 96 | 70 | 94 / 657 |
| 1 | 7 | 7 | 115 | 90 | 100 / 660 |
| 2 | 3 | 9 | 132 | 169 | 95 / 710 |
| 2 | 5 | 25 | 177 | 427 | 96 / 820 |
| 2 | 7 | 49 | 222 | 819 | 122 / 1070 |
| 3 | 3 | 27 | 168 | 593 | 127 / 1123 |

Some inference time for Mamdani and Sugeno type fuzzy controller with MUL-SUM operators, trapezoidal membership functions and COG defuzzyfication (only for the Mamdani) are presented in Table 3. The same remark can be mentioned here, the inference time is small compared to the clock frequency of the microcontroller.

Table 3
Inference time with MUL-SUM operators

| Nr. of input | Nr. of set/input | Nr. of rules | Fuzzification (trapezoidal) [μs] | Inference [μs] | Defuzzyfication Sugeno / COG [μs] |
|---|---|---|---|---|---|
| 1 | 3 | 3 | 78 | 53 | 85 / 910 |
| 1 | 5 | 5 | 96 | 74 | 94 / 918 |
| 1 | 7 | 7 | 115 | 96 | 100 / 922 |
| 2 | 3 | 9 | 132 | 197 | 100 / 1056 |
| 2 | 5 | 25 | 177 | 505 | 101 / 1348 |
| 2 | 7 | 49 | 222 | 971 | 122 / 1903 |
| 3 | 3 | 27 | 168 | 736 | 127 / 1900 |

## Conclusions

In this article the implementation of an embedded fuzzy controller is proposed for 16 bit microcontroller with fast fuzzyfication-inference-defuzzyfication algorithm. It has been demonstrated that because controllers receive information from the process via ADCs and controls the process with the help of DACs the use of DSP based fuzzy controller might not pay the effort for some type of fuzzy controller.

Generally, 16 or only 14 bit DACs are used at the output of a fuzzy controller, in which case the precision of the floating point operation is lost and a DSP based fuzzy controller seems better, faster and more accurate than it is in reality, especially when the state variables of the controlled process change slowly. Thus, the use of a μC based fuzzy controller with integer operations is a reasonable compromise between performance and price.

Simulation and experimental results has been also presented which are supporting the theoretical idea presented in this article.

**Acknowledgement**

This paper was supported by the Hungarian N.Sc. Fund (OTKA No. T 042866) for which the authors express their sincere gratitude.

**References**

[1] Ferenc Farkas, Szilárd Varga, Aleksei Zakharov: Investigation of DC servo drives with fuzzy logic control, Czasopismo Techniczne 4E/1998, Wydawnictwo Politechniki Krakowskiej, Poland, 1998, pp. 35-45

[2] Ferenc Farkas, Aleksei Zakharov, Szilárd Varga: Speed and position controller for DC drives using fuzzy logic, Studies in Applied Electromagnetics and Mechanics (Vol. 16), Applied Electromagnetics and Computational technology II, Amsterdam, Netherlands, IOS Press, 2000, pp. 213-220

[3] Ferenc Farkas: Implementation of fuzzy controller on 16 bit microcontroller, Proceedings of IEEE International Conference on Intelligent Engineering System (INES'99), Stara Lesna, Slovakia, 1-3 November 1999, pp. 567-572

[4] George J. Klir, Bo Yuan: Fuzzy sets and fuzzy logic - Theory and applications, Prentice Hall, New Jersey, 1995

[5] Li-Xin Wang: Adaptive fuzzy systems and control - Design and stability analysis, Prentice Hall, New Jersey, 1994

[6] Retter Gyula: Fuzzy, neurális, genetikus, kaotikus rendszerek – Bevezetés a „lágy számítás" módszereibe, Invest Marketing Bt., Budapest, 2003

[7] Kai Michels: Fuzzy control of electric drive?, European Conference on Power Electronics (EPE'97), Trondheim, Norway, 8-10 Sept, 1997, pp. 1.102-1.106