A METHODOLOGICAL PROPOSAL FOR IMPLEMENTING INTERVAL TYPE-2 FUZZY PROCESSORS OVER DIGITAL SIGNAL CONTROLLERS

Leonardo Leottau Forero[1], Miguel Melgarejo[1]

[1]Laboratory for Automation, Microelectronics and Computational Intelligence
Faculty of Engineering, Universidad Distrital Francisco José de Caldas.
Bogotá D.C., Colombia.
(leottau, mmelgarejo)@ieee.org

Abstract

This article presents a methodological proposal for implementing interval type-2 fuzzy processors over digital signal controller technology. We describe the main considerations that a practitioner or an engineer should follow when implementing an interval type-2 fuzzy system over an embedded processor. These considerations guide the implementation study of eight interval type-2 fuzzy processors, which are fully characterized and tested. Results show that by combining fast computing strategies and technologies like digital signal controllers, the inference time of an embedded type-2 fuzzy processor can be set to hundreds of microseconds.

Key words: Fuzzy logic, Type-2 fuzzy systems, Fuzzy hardware, embedded systems.

1 Introduction

Type-2 Fuzzy Logic Systems (T2-FLS) are rule based systems in which linguistic variables are described by means of Type-2 fuzzy sets (T2-FS) [1, 6]. About a decade of experimental evidence has shown that these systems outperform their Type-1 counterparts (T1-FLS) in applications where non-linearity and uncertainty appear at same time [4]. Up to date, there is not a formal demonstration of that evidence, however more and more works that support it are being developed. Research on T2-FLS is spreading worldwide covering from theory to real world applications [1, 3].

A representation of the inference model for T2-FLS is depicted in Fig. 1 [1]. It begins with fuzzification, which maps crisp points into T2-FS. Next, inference engine computes the rule base by making logical combinations of antecedent T2-FS, whose results are implicated with consequent T2-FS to

form an aggregated output type-2 fuzzy set. Afterwards, Type-Reduction (TR) computes a type-1 fuzzy set that is finally defuzzified in order to obtain a crisp output [3, 18]. The computational complexity of this model is reduced if interval type-2 fuzzy sets are used [6].

Hardware implementation of T1-FLS is a well-known area. Implementations over technologies like Field Programmable Gate Arrays (FPGAs), Digital Signal Processors (DSPs) and microcontrollers of this kind of systems have been reported [2]. Although in recent years, works related to the hardware implementation of T2-FLS have progressively increased, this research area is barely in its beginnings [5, 10, 12, 13, 15, 16, 19]. Complexity of algorithms involved in T2-FLS has mainly limited the implementation of these systems to general purpose computing platforms.

Type-2 fuzzy hardware is a topic of special interest, since the application of T2-FLS to particular fields that demand mobile electronic solutions would be necessary [5]. Some recent applications of T2-FLS have been developed in fields like robotics, communication and control systems among others [5, 16, 17, 20]. It is worth to think about the possibility of embedding T2-FLS handling these applications in order to achieve better communication speeds in smaller areas.

Hardware implementations deal with trade-offs among several variables like area, power consumption and computing speed [2]. Particularly speaking, T2-FLS are complex systems because of the inherent parallel nature of their operations. Besides, these systems pose an additional problem, which is the computation of type-reduction [6, 11, 8, 9]. Thus, type-2 fuzzy hardware is focused on developing architectures, methods and techniques for handling the computational burden of T2-FLS considering the typical restrictions of embedded systems.

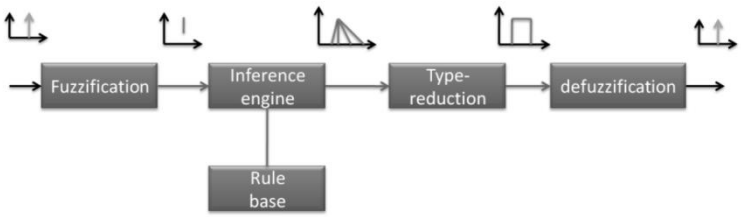


Figure. 1 Type-2 fuzzy system

Day by day, new embedded systems are available in the market. These systems include new capabilities by combining functions found in previous devices. Digital Signal Controller (DSC) technology is one example of it [7]. A DSC is an embedded processor that includes the computing power of a typical Digital Signal Processor mixed with the functionality of a microcontroller.

Implementations of type-2 fuzzy controllers over embedded processors have been presented in [10, 12, 13]. These works have focused on describing particular applications. However, methodological considerations for achieving those implementations are barely mentioned. Thus, practitioners or engineers interested in implementing T2-FLS on hardware are not being fully beneficiated, since key aspects of hardware developments are not entirely described.

This work presents some methodological considerations and strategies for implementing each stage of an interval type-2 fuzzy processor (IT2-FP) over DSC technology. This device combines the parallelism of a DSP with the functionality a microcontroller, achieving an interesting cost–benefit solution for this application. In addition, this work compares several implementations of type-2 fuzzy processors over this technology in order to provide some light about future developments in this area.

The article is organized as follows: Section 2 presents some methodological considerations for implementing interval type-2 fuzzy systems over embedded processors. Section 3 describes an implementation study for eight interval type-2 fuzzy processors over a DSC family. Section 4 presents and discusses several implementation results. Finally, we draw conclusions in section 5.

2 Methodological considerations for implementing interval type-2 fuzzy systems over DSC technology.

This section introduces some strategies for implementing each stage of an Interval Type-2 Fuzzy Processor (IT2-FP) over DSC platforms. Considering the IT2-FP general structure shown in Fig.1, fuzzification, rule base, inference engine, type reduction, and defuzzification are treated as independent blocks. In addition an IT2-FP is defined as N inputs, M Interval Type-2 Fuzzy Sets (IT2-FS) by input, $M_A=N\cdot M$ IT2-FS in the antecedents, $R=M^N$ rules, and M_C IT2-FS in the consequent. Some complementary parameters are considered like discretization of the input universe D_A, discretization of output universe D_C and universe of discourse U.

2.1 Interval type-2 fuzzy sets and footprints of uncertainty

Interval type-2 fuzzy sets are completely characterized by their Footprints of Uncertainty (FOU) [6, 18]. The FOU of an IT2-FS is described by its upper and lower membership functions (MF). Trapezoidal FOU are considered in this work as it is shown in Fig.2. The following computing techniques are focused but not limited to this kind of sets. All procedures are considered assuming a universe within a finite interval [*0, U).*

There are several alternatives to compute the membership grade (MG) of an IT2-FS. Two alternatives for computing MF are presented in [2]: the memory based approach and the function computing method.

2.1.1 Function Computing Approach (FCA)

This method carries out a direct computation of MF by using numeric algorithms that avoid constructing and storing look up tables. It reduces memory usage and facilitates the implementation of MF. However, its execution could require several machine cycles depending on the complexity of MF [2].

Function computing is viable if there are enough resources to execute operations in a defined running time. Handling sets whose MF are described by linear equations is recommended in order to achieve a simpler computation, as in the case of trapezoidal or triangular functions [2, 10, 19].

2.1.2 Memory Based Approach (MBA)

The memory based access approach stores the MGs of every possible input value into a memory. This strategy is executed considerably fast, because it uses the input value as the pointer to the memory and to directly retrieve the MG. However, it is limited by the complexity of the system, because parameters such as: Number of sets, resolution of MG, word length of the processor and level of discretization drastically influence memory consumption [2]. The memory usage of an IT2-FP can be calculated as:

$$Data\_Mem_{mba} = 2 \cdot M_A \cdot D_A + 2 \cdot M_C \cdot D_C \; Words \quad (1)$$

From (1), it can be said that using this method becomes viable if there is enough memory to store all MFs with their respective upper and lower limits.

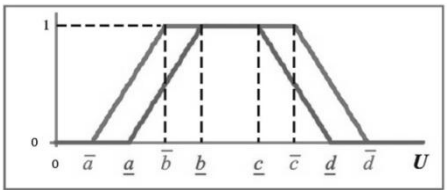


Figure 2. Trapezoidal IT2-FS.

2.2 Fuzzification

Singleton fuzzification is considered in this work [14]. The alternatives proposed in section 2.1 impact directly the fuzzification process. For all cases discussed here, results of fuzzification are stored into a memory array with the following dimension:

$$RAM\_Mem_{fuzz} = 2 \cdot M_A \ Words \tag{2}$$

2.2.1 Fuzzification in the function computing method

For trapezoidal MF, every MG is computed directly by using the expression that represents these functions [2]. Four constants (*a, b, c, d*) are considered as off-line parameters. These are defined in a table with size $2 \cdot 4 \cdot M_A$ and stored in memory. Computation for each boundary is carried out as follows:

$$f(x) = \begin{cases} (x-a)/(b-a), & a \le x < b \\ 1, & b \le x < c \\ (d-x)/(d-c), & c \le x < d \\ 0, & otherwise \end{cases} \tag{3}$$

It is possible to reduce the number of operations by checking first that the input for each set is within the interval *(a, d)*. In this way, unnecessary computing is avoided when the result of fuzzification is zero.

The parameters $1/(b-a)$ and $1/(d-c)$ are the slopes of linear functions in intervals (*a, b*) and (*c, d*). These parameters can be handled as constant off-line values, which are calculated to reduce the number of operations. The constants *(a, b, c, d)* are stored in a three-dimensional array *k_a[i][j][l]*, where the indexes *i, j, l,* respectively refer to *inputs, sets, and limits* (i.e. Upper limit *l=0* or lower limit *l=1*). In order to fuzzify all antecedent sets, three nested loops are used. The internal one to sweep the M_j sets of each input, the middle one to sweep the inputs N_i and the external loop to scan the upper and lower limits L_l of the MF.

2.2.2 Fuzzification in the memory based approach

Fuzzification is reduced to a simple look up memory in this approach [10]. Since computation is not a problem here, memory consumption becomes crucial.

If M_A is the total number of antecedents IT2-FS and D_A is the discretization in the antecedents and upper and lower MF are stored separately, the non-volatile memory consumption is:

$$Data\_Mem_{mbf} = (M_A \cdot D_A)Upper + (M_A \cdot D_A)Lower = 2 \cdot M_A \cdot D_A \ Words \tag{4}$$

From (4), it can be observed that memory usage is proportional to the number of IT2-FS and grows as discretization level increases. Trapezoidal MF are stored in a three-dimensional array *A* with size $N \cdot M \cdot 2 \cdot D_A$.

Just to give a numerical example, if it is intended to work with four T2-FS at the maximum resolution and discretization available in a DSC platform.

$D_A=2^{16}=65536$ discretization levels are obtained in the antecedents. Therefore, a memory space of 1,048,576 bytes is necessary.

2.3 Rule base

Rule base indicates how antecedent sets must be combined and implicated with consequent sets [6]. Thus, a multidimensional matrix sized Min_1 x...x Min_N is provided. This matrix stores the consequent set that corresponds to each rule. The indices of antecedent sets function as pointers for locating a particular consequent set in the matrix.

As an example let's consider a case of $N=2$ inputs:

$$Rule[A_i][B_j] = \begin{bmatrix} C0 & C2 & C1 \\ C3 & C3 & C2 \\ C5 & C4 & C6 \end{bmatrix} \quad (5)$$

Expression (5) implies that there are $M=3$ fuzzy sets $\{A_0, A_1, A_2\}$ at input one as well as in the input two$\{B_0, B_1, B_2\}$. There are $R=9$ rules $\{r_0, \ldots, r_8\}$ and $M_C=7$ fuzzy sets in the consequent $\{C_0, \ldots, C_6\}$. E.g. the rule number five is:

$$\text{"IF } X \text{ is } A_1 \text{ and } Y \text{ is } B_2 \text{ then } Z \text{ is } C2\text{"}. \quad (6)$$

In the example, there are less consequent sets than rules because one or more rules have the same consequent set. Note that rules one (A_0,B_1) and five (A_1,B_2) are sharing consequent $C2$, whereas rules three (A_1,B_0) and four (A_1,B_1) are sharing consequent $C3$.

2.4 Inference Engine

2.4.1 Mamdani inference

Mamdani inference [6, 14] is selected, because it is one of the most suitable methods for real time hardware operation due to its simplicity [2]. This method uses MIN T-Norm and MAX S-Norm as implication and aggregation operators respectively. In Mamdani implication a rule collection is given as shown below:

$$Rule\ r\text{: } If\ X_1\ is\ A_1^r\ and\ X_2\ is\ A_2^r\ and \ldots X_N\ is\ A_N^r\ then\ Y\ is\ C^r \quad (7)$$

Where $r = 1, 2, \ldots, R$

The inferred conclusion is the active rule collection combination given by (8). The activation grade of rule r is given by (9).

$$\mu'_C(y) = \max_r(min\ (\alpha^r, \mu_C^r(y))) \quad (8)$$

$$\alpha^r(y) = min\,(\mu^r_{A_1}(x_1), \ldots \mu^r_{A_n}(x_n)), \tag{9}$$

where $\mu^r_{A_1}(x_1), \ldots \mu^r_{A_n}(x_n)$ are the MFs sets of the inputs $x_1 \ldots x_n$ and $\mu^r_C(y)$ are the consequent MFs, in both cases for rule *r*.

2.4.2 Computing Mamdani inference

An algorithm to compute the inference engine over sequential platforms is set out in [2] and it is adjusted for T2-FLS in [10]. A modification of this algorithm is proposed here. The activation grades in the antecedents (9) are computed with a loop that is executed before the rest of the inference engine (8), since results of antecedents combinations are constant values along their respective consequent sets. The modified algorithm is presented as follows:

```
Inference Engine Algorithm
{
    For rule r=1: rule R
    Amin[r] = min[AMVs[r]]; Obtaining the minimum of the
                            antecedent membership values (AMV)
    For i=1: Dc
    {f[i]= 0;
       For rule r=1: rule R
       {Aux = CMV[Rule[r]][i];   Obtaining the consequent
                                 membership value (CMV)
        Aux = Min[Aux, Amin[r]];
        f[i] = Max[Aux, f[i]];
       }
    }
}
```

The algorithm uses two loops. The first one is used to obtain the antecedent of all rules. It computes the minimum among the fuzzified inputs in each rule. Results are stored into a table with size $2 \cdot R$ located in memory using two arrays *Amin_u[R]* for the upper boundaries and *Amin_l[R]* for the lower ones. The latter computes (8) so that the minimum between (9) and the MG of the consequent set for rule *r* is found. This process is executed for all rules while aggregation is being computed by sweeping the whole universe. Since this algorithm is proposed for IT2-FP, it must be implemented for upper and lower boundaries.

In contrast to fuzzification, where a unique value is obtained from MF, the entire universe must be swept for active consequent T2-FS in the inference engine. Despite of this, it is also possible to apply the same techniques proposed in section 2.1 in the inference engine. So, the inferred conclusion per rule is an IT2-FS, which is stored in memory as an array with size $2 \cdot D_C$.

2.4.3 Function computing approach for inference engine

Since trapezoidal MFs are used, every crisp value is computed using (3). Parameters of (3) are loaded off-line into a two-dimensional array $k_C[j][l]$ with size $2 \cdot 4 \cdot M_C$. Where the indexes *j, l*, respectively refer to *consequent sets* and *limit*. This inference engine model is executed as it is described in previous subsection, but obtaining each consequent membership value $\mu_C^r(y)$ in (8) by using (3).

In order to speed up the computation of this inference engine method, the same strategy introduced in section 2.2.1 can be used here. If the pointer is not within the interval *(a, d)*, the MG is zero, so the inference engine computation can be avoided for this point.

2.4.4 Memory based approach for inference engine

If both MF limits are stored individually and assuming that M_C is the number of consequent sets with a word length of *W* bits, the data memory consumption is as follows:

$$\text{Data\_Mem}_{\text{mbi}} = (M_C \cdot D_C)\text{Upper} + (M_C \cdot D_C)\text{Lower} = 2 \cdot M_C \cdot D_C \text{ Words} \quad (10)$$

MF are stored in tables, which are handled as three-dimensional arrays with size $2 \cdot M_C \cdot D_C$ treated as *C[j][i][l]*, where *j*, *i* and *l* refers to consequent set M_{Cj}, upper limit *l=0* or lower limit *l=1* and stored membership grade MG_i respectively. This inference engine model is executed as it is described in subsection 2.4.2, obtaining each consequent membership value $\mu_C^r(y)$ in (8) reading the correspondent array *C*.

2.5 Type Reduction and Defuzzification

There are several type-reduction alternatives such as: centroid, center of sums, center of sets, height and modified height. Depending on the method, there is a compromise between accuracy and computational complexity. Centroid type-reduction is the most accurate method because it uses the union of whole output sets and not just singletons. In contrast, it is the most computational expensive [6]. Thus, this type-reducer is chosen in order to obtain the best accurate output while the hardware platform is forced to the highest computational effort.

As it is mentioned in section 2.1, the universe is set in the interval [*0, U)*. In addition a linear discretization scheme with equidistant points is proposed in order to reduce the computational complexity of the algorithms involved in type-reduction. Thus, a particular point within the universe can be computed as [10]:

$$x = k \cdot U/Dc \tag{11}$$

It implies that the execution of some divisions will be required to find *x*. It is proposed to compute off-line $U\_Dc = U/Dc$, in order to reduce the amount of operations, particularly avoiding divisions. On the other hand, if the size of the universe *U* is equivalent to the discretization levels in the consequent *Dc*, the computing complexity is reduced because *x=k*.

2.5.1 Enhanced Karnik Mendel algorithm(EKM)

The EKM algorithm proposed in [11] uses statistically defined values as initialization points to reduce the amount of iterations that are necessary to converge [6]. The algorithm verifies the stop condition before carrying out a new computation. Therefore, one iteration is saved.

Let be $\overline{f}_i$ and $\underline{f}_i$ respectively the upper and lower membership grades of an inferred interval type-2 fuzzy set and x_i the points of the universe of discourse. Therefore, the EKM algorithm for computing the minimum limit c_l of the centroid of the set is presented as follows:

i. Set:

$$k = round(N/2.4) \tag{12}$$

$$D = \sum_{i=1}^{k} x_i \overline{f}_i + \sum_{i=k+1}^{Dc} x_i \underline{f}_i \tag{13}$$

$$P = \sum_{i=1}^{k} \overline{f}_i + \sum_{i=k+1}^{Dc} \underline{f}_i \tag{14}$$

$$y = D/P \tag{15}$$

ii. Find $k' \in [1,N-1]$ such that $x_{k'} \leq y \leq x_{k'+1}$

iii. Check if $k'=k$. If yes, stop, set $\boldsymbol{c_l = y}$ and call $L = k$. Else, continue

iv. Compute s=sign(k'- k), and

$$D' = D + s \sum_{i=min(k,k')+1}^{max(k,k')} x_i (\overline{f}_i - \underline{f}_i) \tag{16}$$

$$P' = P + s \sum_{i=min(k,k')+1}^{max(k,k')} (\overline{f}_i - \underline{f}_i) \tag{17}$$

$$y = D'/P' \tag{18}$$

v. Set y = y', D = D', P = P', k = k'. Go to step ii.

The observation introduced at the beginning of this subsection regarding the linear discretization of the output universe is used in order to find the value of *k'*. So, the following computation is applied:

$$k' = y \cdot Dc/U \tag{19}$$

At the beginning of each limit procedure, is necessary to calculate the rounding of *k*, using eq.(12). In order to reduce the quantity of instructions, it is possible to set this rounding off-line and to take it as a constant whenever it is required. The algorithm is executed using a *While – Do* loop. A fixed flag is the condition that forces repeatedly the execution of the iteration within the loop until the break condition is reached.

A similar procedure is carried out for computing the maximum limit c_r of the centroid of the IT2-FS [11]. Thus, two independent loops must be implemented to obtain the type-reduced set before defuzzification.

2.5.2 Improved Iterative Algorithm with Stop Condition (IASCO)

The recursive algorithm proposed in [8,9] is used to find the upper and lower boundaries of the centroid (*cl, cr*) of an IT2-FS without looking for the switching points L and R. On the other hand, it combines exhaustive search and iterations, which progressively increase *L* and *R* until the optimal point is found. The stop condition of this algorithm is derived from the properties of the centroid function. The algorithm for computing c_l is presented as follows:

i. Initialization:

$$D_0 = \sum_{i=1}^{Dc} x_i \underline{f}_i \tag{20}$$

$$P_0 = \sum_{i=1}^{Dc} \underline{f}_i \tag{21}$$

$$C_{min} = x_N \tag{22}$$

ii. Start k=0, increase it as k=k+1 and execute ii – iii

$$D_k = D_{k-1} + x_k \left[\overline{f}_k - \underline{f}_k\right] \tag{23}$$

$$P_k = P_{k-1} + \left[\overline{f}_k - \underline{f}_k\right] \tag{24}$$

$$c_l(k) = \frac{D_k}{P_k} \tag{25}$$

$$\textit{iii. If} \quad cl \leq cmin \quad \textit{then } cmin = cl(k). \quad \textit{Else stop computations and set } cl = cmin \tag{26}$$

The variable k is used as a pointer. Parameter x is directly related to k regarding (11). The upper and lower limits of the inferred set are two vectors, which are accessed using the pointer k. Results from intermediate expressions (23)-(25) are handled as crisp values, which are updated iteration after iteration. Thus, using few registers as temporal variables is enough. Two independent searching loops are implemented for c_l and c_r.

2.5.3 Defuzzification

Once type-reduction has found the limits of the centroid, defuzzification computes the mean between c_l and c_r to obtain the crisp output of the IT2-FP [6]. No division is required here since it can be replaced by a shift-right operation, which is faster in specialized embedded processors like DSCs.

3 Implementation study

This section introduces some aspects about the hardware implementation of the models proposed in the previous section. Characteristics of processors to be implemented are previously defined as well as the available hardware resources of the DSC technology.

3.1 Implementation parameters

This implementation study is done based on a system with the following characteristics: Two inputs, three type-2 interval fuzzy sets by input, nine rules, one interval type-2 fuzzy set in the consequent by each rule and D_A and Dc discretization levels for antecedent and consequent universes respectively. The IT2-FS used and its parameters are shown in Fig.3 and Table 1. These are chosen in this way in order to cover the whole universe. The universe is set between $[0, U)$, where U is limited to a thousand points.

Implementation is developed in C language. Therefore, maximum and minimum functions are carried out using *if then* sentences to compare and to determine the greater value for *MAX* function or the smaller one for *MIN* function. Besides, index and pointers of tables, arrays, vectors and matrix, start from position zero.

Since this is a two-input system with three fuzzy sets by input, a 3x3 square matrix is set. For this system the following matrix is implemented:

$$Rule[A_i][B_j] = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \tag{27}$$

Then, the matrix implies the following rule collection:

$$\begin{aligned} &Rule\ 0\text{: } if\ in0\ is\ A_0^0\ \ y\ \ in1\ is\ B_0^0\ then\ C^0 \\ &Rule\ 1\text{: } if\ in0\ is\ A_0^1\ \ y\ \ in1\ is\ B_1^1\ \ then\ C^1 \\ &Rule\ 2\text{: } if\ in0\ is\ A_0^2\ \ y\ \ in1\ is\ B_2^2\ \ then\ C^2 \\ &Rule\ 3\text{: } if\ in0\ is\ A_1^3\ \ y\ \ in1\ is\ B_0^3\ \ then C^3 \\ &\qquad\vdots \\ &Rule\ 8\text{: } si\ IN0\ is\ A_2^8\ \ y\ \ IN1\ is\ B_2^8\ then\ C^8 \end{aligned} \tag{28}$$

Table 1. Parameters per IT2-FSs shown in Fig.3.

	A0		***A1***		***A2***		***B0***		***B1***		***B2***	
	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$
a_a	-304	-256	160	208	624	672	-416	-320	64	160	544	640
b_a	-48	0	416	464	880	928	-128	-32	352	448	832	928
c_a	208	160	672	624	1136	1088	160	64	640	544	1120	1024
d_a	464	416	928	880	1392	1344	448	352	928	832	1408	1312

	C0		***C1***		***C2***		***C3***		***C4***		***C5***		***C6***		***C7***		***C8***	
	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$	$\overline{f}$	$\underline{f}$
a_c	-112	-96	16	32	144	160	272	288	400	416	528	544	656	672	784	800	912	928
b_c	-32	-16	96	112	224	240	352	368	480	496	608	624	736	752	864	880	992	1008
c_c	32	16	160	144	288	272	416	400	544	528	672	656	800	784	928	912	1056	1040
d_c	112	96	240	224	368	352	496	480	624	608	752	736	880	864	1008	992	1136	1120

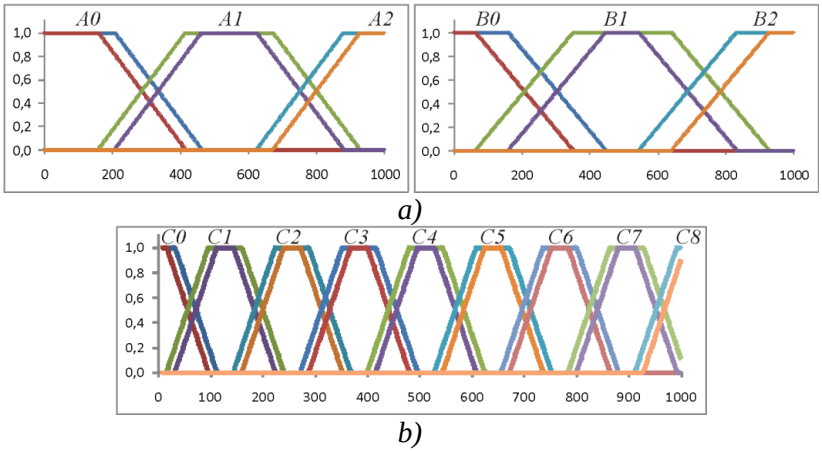


Figure 3. Type-2 Fuzzy sets used: a) Antecedents, and b) Consequent.

3.2 Performance indices

Three performance indices are defined in this implementation study. These are:

Memory usage: the memory consumption per processor stage is computed from (2), (4) and (10).

Instructions (Inst): Since the code is written in C language, this parameter is different from the amount of assembly instructions. This parameter exclusively refers to the amount of code instructions demanded by an implemented algorithm.

Clock Cycles (ClkC): This parameter refers to the amount of clock cycles employed by a processor to execute an algorithm. To obtain the time (in seconds) demanded by any process, clock cycles must be divided by the frequency of processor´s clock as follows:

$$Processor\_Time = ClkC/fclk \tag{29}$$

Absolute error: In order to obtain this parameter, a Microsoft ™ EXCEL® based model with *Dc=1000* is used as reference. The absolute error is given by:

$$AE = |Vref - Vobt|, \tag{30}$$

where *Vref* is the crisp output of the type-2 fuzzy processor in EXCEL® and *Vobt* is the crisp output obtained from the processor implemented over DSC technology.

Table 2. Maximum and minimum available hardware resources of DSP56800E family

	From	Chip	Up to	Chip
Clock frequency	32Mhz	56F80XX	120Mhz	5685X
Program memory	12 KB	56F8011	512 KB	56F836X - 56F816X
RAM memory	1K x 16-bit	56F8011	40K x 16-bit	56858
Data memory	2K x 16-bit	56F801	16K x 16-bit	56F836X
Word length	16 bits			

3.3 Hardware Resources

Implementation is carried out over the Freescale™ DSP56800E® family. It provides low-cost, low-power, mid-performance computing, combining DSP signal processing power and parallelism, microcontroller functionality and several flexible integrated peripherals. The DSP56800E architecture is based on the parallel execution of three operative units: A data arithmetic logic unit (ALU), an address generation unit (AGU) and a program controller [7]. The available hardware resources of this DSC family are shown in Table 2.

3.4 Tests

Two different strategies for fuzzification, inference engine and type-reduction are considered. Eight possible combinations of different processors are obtained as it is described in Table 3. Each processor is validated first using a Microsoft™ Excel® model. Then, it is programmed in C Code, simulated and debugged by means of the CodeWarrior® software suite. Finally, the processor is loaded on the DSC platform.

The performance indices of each processor are obtained as follows:

1. Set the code on the simulation and on-chip debugging software tool CodeWarrior® v.8.23.
2. Run one complete inference and register separately the number of instructions and clock cycles demanded by the fuzzyfier, inference engine and output processor.
3. Record the output value obtained.
4. Compute the absolute error using eq. (30)
5. Repeat 10 times steps 2 – 4 with different input values.
6. Find the average and standard deviation of the instructions, clock cycles and the absolute error.
7. Repeat the procedure for the following discretization levels: $D_C=10$; $D_C=100$; $D_C=1000$.

Table 3. Eight possible processors

	Fuzzyfier	Inference Engine	Type reducer
Processor 1	Function based	Memory based	EKM
Processor 2	Function based	Memory based	IASCO
Processor 3	Memory based	Memory based	EKM
Processor 4	Memory based	Memory based	IASCO
Processor 5	Function based	Function based	EKM
Processor 6	Function based	Function based	IASCO
Processor 7	Memory based	Function based	EKM
Processor 8	Memory based	Function based	IASCO

4 Implementation results and discussion

4.1 Implementation results

In this section, implementation results are presented in Tables 4-8 and Figures 4-6. Memory usage of each strategy implemented per stage of the IT2-FP is presented in Table 4 for fuzzyfiers and rule collection and Table 5 for inference engine. Besides, Figure 4 presents the total memory consumption of the eight IT2-FP. Results of clock cycles and instructions demanded by each strategy are presented in Table 6 for fuzzyfiers, Table 7 for inference engines and Table 8 for output processors. In addition, Figure 5 presents the total clock cycles and instructions demanded by the eight processors. Finally, Figure 6 presents the absolute error of eight implemented processors.

4.2 Discussion

It can be observed about fuzzification that data memory usage in FCA is smaller than data memory usage in MBA. The FCA uses about 0.4% of the MBA resources. However, MBA fuzzyfier is about 2.5 times faster than FCA based fuzzyfier.

Data memory usage of MBA inference engine increases linearly with discretization levels in the consequent *Dc*. On the contrary, the FCA memory usage is constant and it is independent of the discretization levels in the output universe.

FCA is almost as fast as MBA inference engine. FCA is about 0.3% slower for *Dc*=10, 4% slower for *Dc*=100 and about 5% slower for *Dc*=1000. Theoretically, the MBA is faster than FCA [2]. However, the implementation strategy proposed in section 2.4.3 speeds up FCA inference engine. Saving computations when MFs are zero compensates the operations demanded by computing equation (3). However, the data memory usage in FCA is about 20% of the MBA for *Dc*=10, 2% for *Dc*=100 and about 0.2% for *Dc*=1000.

EKM algorithm is about 1.4 times faster than IASCO for *Dc*=10 and about 2.2 times faster for *Dc*=1000. It is especially interesting because [8] shows that IASCO outperforms the EKM algorithm in general purpose microprocessors. This difference can be explained because the first one executes more divisions than the latter. The DSC technology used in this implementation study does not include any dedicated divider. So, divisions are computed by means of software subroutines, which are typically slower than dedicated hardware units.

According to Table 2 and Fig 5(a), an Interval type-2 fuzzy processor like the one considered in this study would exhibit a global inference time between 139.6 μs and 5.89 ms. These times are obtained from the fastest processor (i.e. *Dc* = 10, MBA in fuzzification, MBA in inference engine and EKM type-reduction) and the slowest processor (i.e. *Dc*=1000, FCA in fuzzification, FCA in inference engine and IASCO type-reduction) respectively. Since inference times are in the scale of hundred of microseconds and milliseconds, these processors can be used as fuzzy controllers in control applications [14, 12, 17].

Other works have reported implementations of type-2 fuzzy systems over embedded processors. Coupland et al [13] implemented a general type-2 fuzzy controller with nine rules over a Microchip(c) PIC24F platform achieving an inference time of 306 ms. Besides, Bulla et al [10] developed an interval type-2 fuzzy system with 4 rules over a Freescale MC68HC908AP32 whose inference time is about 34.29 ms. The results obtained in this study show that inference time for type-2 fuzzy systems can be set in the order of microseconds by using a combination of faster technologies and computing strategies.

From Figure 6, it can be said that processors implemented with FCA fuzzyfier generate higher error than processors with MBA fuzzyfier. The average among absolute errors of the three discretization levels of processors P1, P5 and P6 is about 5.6. These processors are the less accurate regarding the software reference. This measure is about 5.7% greater than the obtained by processor P7, which is the most accurate.

Table 4. Results for memory usage of fuzzyfiers ad rule collection.

Fuzzyfier	Data memory (Bytes)	RAM (Bytes)
Memory based approach	24000	24
Function computing approach	96	24
Rule collection	18	36

Table 5. Results for memory usage of inference engine

	Memory based approach		**Function computing approach**	
Dc	Data memory (Bytes)	RAM memory (Bytes)	Data memory (Bytes)	RAM memory (Bytes)
10	360	40	72	40
100	3600	400	72	400
1000	36000	4000	72	4000

Table 6. Results of clock cycles and instructions demanded by fuzzyfier

Fuzzyfier	**ClkC.**	**Desv. ClkC.**	**Instr.**	**Desv. Instr.**
Memory based approach	883.20	2.53	498.70	0.95
Function comp. approach	2215.30	301.42	1203.90	177.06

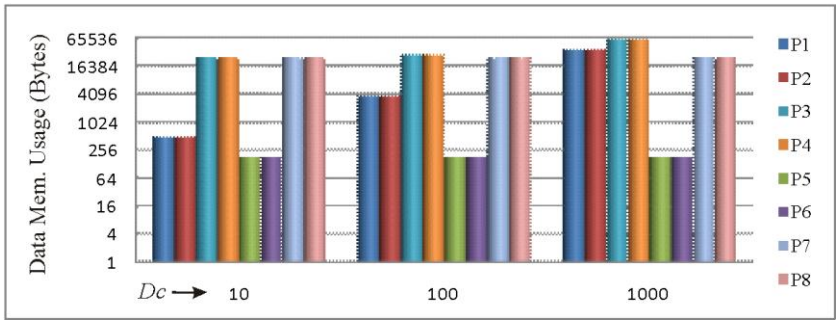
Table 7. Results of clock cycles and instructions demanded by inference engine.

	Dc	**ClkC**	**Desv. ClkC**	**Instr.**	**Desv. Instr.**
Memory based approach	**10**	13078.90	31.54	6274.40	26.90
	100	117468.30	220.31	55182.80	215.30
	1000	1153346.30	2174.70	539274.80	2143.98
Function comp. approach	**10**	13120.70	29.97	6488.20	25.98
	100	122374.10	213.42	59507.60	211.56
	1000	1213775.40	2084.25	587190.40	7550.41

Table 8. Results of Clock cycles and instructions demanded by type-reduction

	Dc	**ClkC**	**Desv. ClkC**	**Instr.**	**Desv. Instr.**
EKM	**10**	2798.50	440.97	1732.50	278.99
	100	18379.90	1887.29	10709.50	1093.83
	1000	138733.00	15146.48	76699.70	8457.46
IASCO	**10**	4003.60	1199.25	2768.80	879.11
	100	34327.60	11755.05	23638.30	8617.05
	1000	315129.40	111793.85	217500.50	82348.15

a)



b)

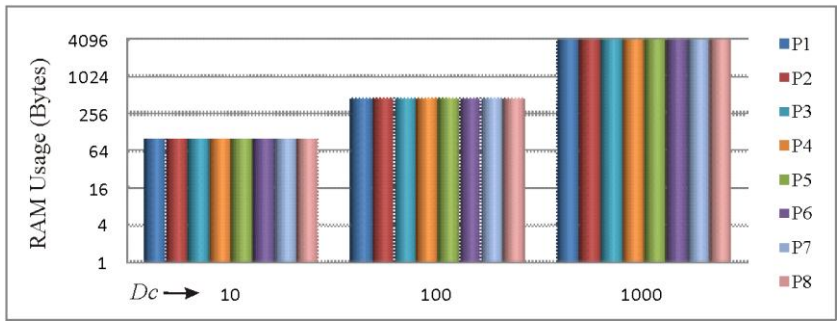
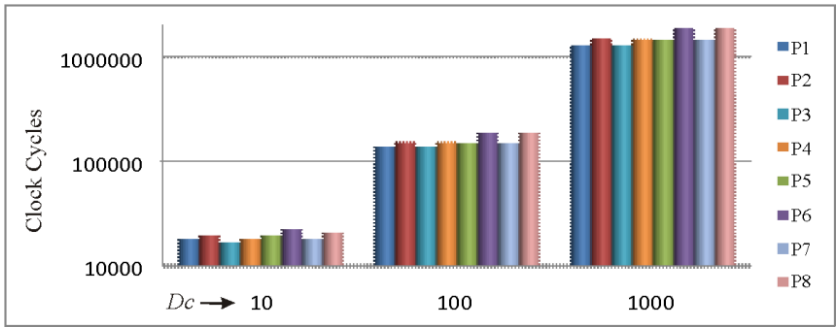


Figure 4. Memory usage for the eight processors: a) Data Memory, and b) RAM

a)



b)

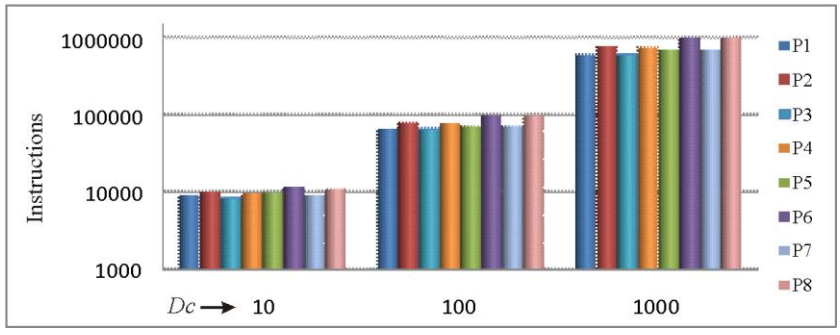


Figure 5. For the eight processors: a) Clock cycles, and b) Instructions.

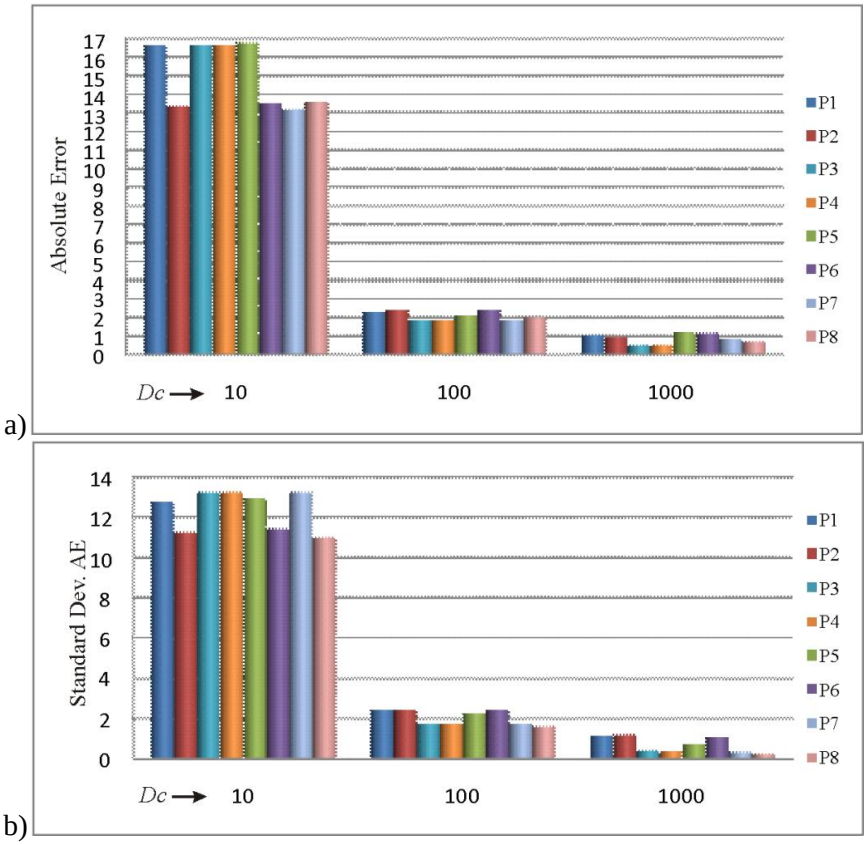


Figure 6. For the eight processors: a) Absolute Error, and b) Standard deviation of AE.

5 Conclusions

Some methodological considerations for implementing interval type-2 fuzzy processors over DSC technology have been described. These allowed implementing eight interval type-2 fuzzy processors considering three different levels of discretization in the consequent. The processors were fully characterized and tested. Results show that the inference time of a type-2 fuzzy system running over an embedded processor can be set in the order of microseconds by combining fast technologies like DSC and computing strategies.

Several possibilities for implementing IT2-FLS based on two strategies for fuzzification, inference engine and type-reduction can be derived directly from this work. Thus, a practitioner or an engineer can choose one of these

regarding some parameters of the target application like inference time, accuracy, resolution and available computing and memory resources.

Since this work is an introduction to the implementation of IT2-FP over DSC platforms, possibilities to improve the parameters of a type-2 processor, such as inference time and memory resource are open. Besides, methodological aspects proposed in this work could be used as a reference to carry out implementations of IT2-FP over more powerful DSP platforms.

References

1. Mendel J. M., 2007, *Advances In Type-2 Fuzzy Sets and Systems*, Information Sciences, 177,1, pp. 84-110.
2. Baturone I. et al, 2000, *Microelectronics Design Of Fuzzy Logic-Based Systems*, CRC Press LLC, London.
3. Mendel J. M., 2007, *Type-2 Fuzzy Sets and Systems: An Overview*, IEEE Computational Intelligence Magazine, 2,1, pp. 20–29.
4. John R. and Coupland S., 2007, *Type-2 Fuzzy Logic A Historical View*, IEEE Computational Intelligence Magazine, 2,1, pp. 57-62.
5. Melgarejo M., Pena-Reyes C. A., 2007, *Implementing Interval Type-2 Fuzzy Processors*, IEEE Computational Intelligence Magazine, 2,1, pp. 63-71.
6. Mendel J. M., 2000, *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*, Prentice Hall, New Jersey.
7. Freescale Semiconductor, Http://www.Freescale.Com/Dsc, Digital Signal Controller 56800/E Reference. Revised on jan-2010.
8. Duran K., Bernal H. and Melgarejo M., 2008, *Improved Iterative Algorithm For Computing The Generalized Centroid Of An Interval Type-2 Fuzzy Set*, Proceedings of NAFIPS 2008, New York City, pp. 1-5.
9. Melgarejo M., 2007, *A Fast Recursive Method to Compute the Generalized Centroid of an Interval Type-2 Fuzzy Set*, Proceedings of NAFIPS 2007, San Diego, California, pp. 190 – 194.
10. Bulla J., Sierra G. and Melgarejo M., 2008, *Implementing A Simple Microcontroller-Based Interval Type-2 Fuzzy Processor*, Proceedings of the IEEE 51th Middle West symposium on Circuits and Systems International Symposium, Knoxville, TN, pp. 69-72.
11. Dongrui W., Mendel J., 2009, *Enhanced Karnik-Mendel Algorithms*, IEEE Transactions on Fuzzy Systems, 17, pp. 923-934.
12. Lynch C., Hagras H. and Callaghan V., 2007, *Parallel Type-2 Fuzzy Logic Co-Processors for Engine Management*, Proceedings of the 2007 IEEE International Conference on Fuzzy Systems, London, UK, pp. 907-912.
13. Coupland S., Wheeler J. and Gongora M., 2008, *A Generalised Type-2 Fuzzy Logic System Embedded Board and Integrated Development Environment*, Pro-

ceedings of the 2008 IEEE International Conference on Fuzzy Systems, Hong Kong, China, pp. 681-687.

14. Wang L. X., 1997, *A Course in Fuzzy Systems And Control*, Prentice Hall, New Jersey.
15. Sepulveda R., et al, 2009, *Modeling and Simulation of the Defuzzification Stage of a Type-2 Fuzzy Controller Using the Xilinx System Generator and Simulink*, In: Evolutionary Design of Intelligent Systems, (Eds), SCI 257, Springer-Verlag, Berlin Heidelberg, pp. 309–325.
16. Melgarejo M., Garcia A. and Pena-Reyes C., 2004, *Pro-two: a hardware based platform for real-time Type-2 fuzzy inference*, Proceedings of the 2004 IEEE International conference on Fuzzy Systems, Budapest, Hungry, pp. 977-982.
17. Hagras H., 2007, *Type-2 FLC's: a New Generation of Fuzzy Controllers*, IEEE Computational Intelligence Magazine, February 2007, pp. 30-43.
18. Karnik N. and Mendel J., 2001, *Centroid of a Type-2 Fuzzy Set*, Information Sciences, vol. 132, pp. 195-220.
19. Leottau L., Melgarejo M., 2010, *Implementing an Interval Type-2 Fuzzy Processor onto a DSC 56F8013*, Proceedings of the 2010 IEEE International conference on Fuzzy Systems, (in press).
20. Castro J., Castillo O., Melin P. ,2007, *An Interval Type-2 Fuzzy Logic Toolbox for Control Applications*, Proceedings of the 2007 IEEE International conference on Fuzzy Systems, London, UK, pp. 1-6.