

DSP-BASED FAST FUZZY LOGIC CONTROLLERS

I. Kalaykov, B. Iliev

Örebro University
Department of Technology
70182 Örebro, Sweden
Tel: +461930 3625, Fax: +461930 3463
e-mail: {ikv,biv}@tech.oru.se

R. Tervo

University of New Brunswick
Dept. of Electrical and Computer Engineering
Fredericton, NB, Canada E3B 5A3
Tel: (506) 453-4561, Fax: (506) 453-3589
e-mail: tervo@unb.ca

Abstract

This paper presents an approach to implementing a fast FLC on a DSP in order to achieve high speed operation with a small sampling interval. This approach uses some specific techniques for hardware-software co-design, where hardware-specific properties of DSP architecture are utilized to reduce the number of computations required in software. The DSP approach is cost-effective and useful for processes with small time-constants.

Introduction

Fuzzy control techniques have attracted significant interest and have become an important part of modern control engineering. The use of linguistic knowledge in the form of IF-THEN rules gives a fuzzy system the ability to work as a universal approximator to nonlinear functions. A typical fuzzy logic controller (FLC) consists of a fuzzification module, fuzzy inference engine, defuzzification module (see Fig. 1) and pre- and post-processing modules. A fast FLC must be used when the process to be controlled has relatively fast dynamics. Fuzzy logic hardware can be based on either analog or digital technology. Analog fuzzy elements use the nonlinear characteristics of active devices; hence they are implemented in very simple structures with a small design area. Ultimately, their speed is limited by the semiconductor technology. In contrast, digital technology offers more flexibility and higher speed but requires more design area and limits the number of inputs and outputs. Alternatively, a low cost FLC can be implemented with a general-purpose 8/16-bit microcontroller; however, the sampling interval is necessarily bigger than with a direct hardware implementation. The paper presents an approach to implementing a fast FLC on a DSP that is both cost-effective and useful for fast processes. Some earlier work in this area was done, e.g. by George (1992) who implemented a fuzzy logic compensator on a TMS320C14 DSP based servo motor control system and Kumbala, Akbarzadeh and Jamshidi (1995) on TMS320 DSP based neuro-fuzzy controller. Later George (1996)

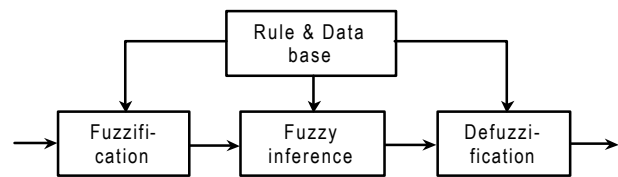


Fig. 1. General fuzzy logic controller

and Del Campo and co-workers (1998) generalized that DSPs are a cost-effective, complete flexibility and high-performance alternative to microprocessor or microcontroller embedded systems. A CAD design tool for FLCs on a DSP was reported by Del Campo and Tarela (1998). However, in all these references the popular FLC algorithm of Mamdani is used, where all FLC modules are treated conventionally employing all necessary arithmetical operations that slow down the performance. One very useful and simple approach to accelerate FLC performance is to employ the fired-rules-hyper-cube (FRHC) concept developed by Kalaykov (1998) and (1999). The approach significantly reduces the search for rules that contribute to the FLC output, as will be described later. The FRHC assumes the input membership functions (MFs) are defined in such way that not more than two of them overlap at any point of the input space.

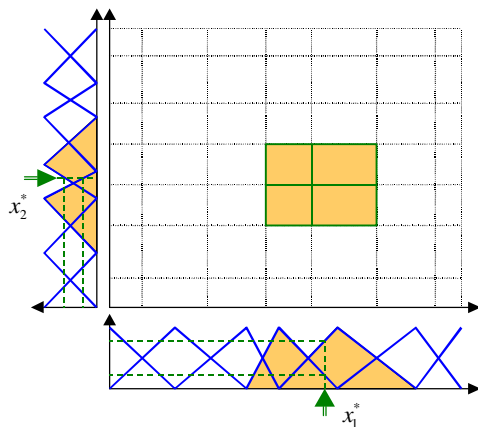


Fig. 2. Fired-Rules-Hyper-Cube (FRHC) concept

This is not a real limitation, as in most applications the uncertainty, expressed by fuzzy sets, is on the boundary between two fuzzy sets. An example is given in Fig. 2 for a two-input-one-output controller. When the crisp input variables vary (e.g. take values x_1^* and x_2^*) they fire only two fuzzy sets, as highlighted in Fig. 2. In the general case, a maximum of N_{over}^m rules can be activated to form the FRHC, where N_{over} is the number of overlapping MFs and m is the number of inputs. As the illustration shows, the FRHC can have at most four active rules.

To achieve the main goal of FLC operation at high speed, some specific techniques for hardware-software co-design are applied, where some hardware-specific properties of a DSP architecture are utilized to reduce the number of computations required in software. For example, defuzzification which needs scalar multiplication of vectors plus addition of scalars and division can be performed by typical DSP instructions, as these same operations are difficult for a direct hardware design. At the same time, other operations related to defuzzification and fuzzy inference require more simple circuits, so there is a need only for simple interfacing software to complete the hardware-software co-design. Details are presented in the next section, followed by a brief description of the design procedure in a MATLAB environment. Examples and comparisons for several DSP platforms and comments conclude the paper.

Fast fuzzy logic controller on DSP

The data structures which are to be stored in memory are described, followed by the principle of operation of the FRHC algorithm in reducing the number of computations. The universe of discourse (UD) of each input is sampled, depending on the resolution of the analog-to-digital converter (ADC) measuring the input analog signals (e.g. for an 8-bit ADC, 256 points have to be considered). For each defined fuzzy set the respective degree of membership is obtained and stored in a dedicated structure of memory blocks. First, an index is assigned to each one of the 2^N points corresponding to the label of the fuzzy set that is fired (activated) at this point. To do this, the symbolic names of the fuzzy sets are replaced by numerical indices, as shown in the illustrative table below.

Symbolic names	NB	NS	Z	PS	PB	...
Numeric indices	0	1	2	3	4	

As it is assumed that at each point there can be no more than two MFs which can overlap (always adjacent odd-indexed and even-indexed MFs) the entire data set is divided into two parts - one for odd- and one for even-indexed fuzzy sets (the upper half and the lower half of Fig. 3, respectively). When a crisp input value enters the support of any MF, the respective degree of membership can be retrieved directly from one of the memory blocks “mem₀” or “mem₂”, while the index (label) of the fired fuzzy set is retrieved from “mem₁” or “mem₃”. In this way, computations are avoided and at the same time the user has the option to define an arbitrary form of MF, the only requirement being that not more than two MFs overlap. The total execution time of fuzzification is then equal to $t_{FUZ} = m * 4 * (t_{RD} + t_{WR})$, where t_{RD} is the time for reading and t_{WR} is the time for writing to memory and m is the number of FLC inputs. An example of the memory allocation for two FLC inputs with a 10-bit ADC is illustrated in Fig. 4, where the code \$F is used to denote an undefined MF. If memory is limited, an option to reduce the table is to store the fuzzy sets indices from “mem₁” and “mem₃” as lower and upper part of 8-bit words and to apply separation whenever necessary, as was made in the current implementation.

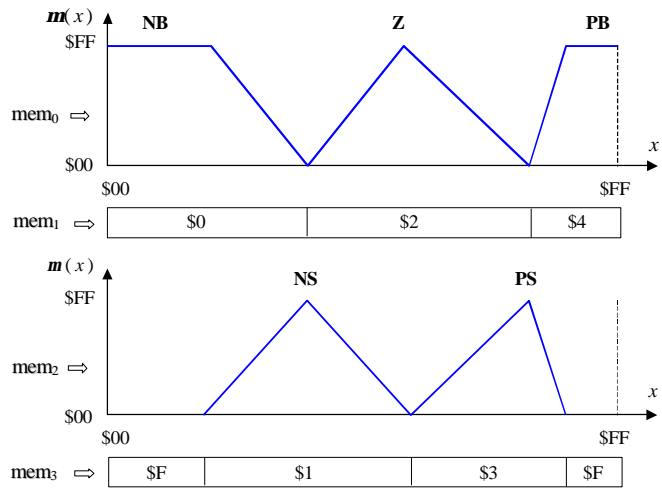


Fig. 3. Description of membership functions

The rule-base is stored in a dedicated memory block, which has to be accessed readily. The values must be in a form to be accessed correctly using the introduced indexing system for the fired MFs. The rules are “IF x_1 is z_i AND x_2 is z_j THEN $y = c_k$ ”, where c_k is an integer from 0 to 255 (assumes an 8-bit DAC converter). If a rule is not defined for some x_1 and x_2 , the respective c_k takes a value of zero. The rules are ordered by combining the first (leftmost) fuzzy set on x_1 gradually with all sets on x_2 , starting also from the leftmost fuzzy set on x_2 . When all sets on x_2 are combined, the operation continues with the next set on x_1 in the same way until all combinations are covered. A rule base example for five fuzzy sets {NB, NS, Z, PS and PB} defined on x_1 and x_2 is given in Table 1.

Table 1. Rule base for two-input controller

IF x_1 is NB AND x_2 is NB THEN $y = c_1$
IF x_1 is NB AND x_2 is NS THEN $y = c_2$
IF x_1 is NB AND x_2 is Z THEN $y = c_3$
IF x_1 is NB AND x_2 is PS THEN $y = c_4$
IF x_1 is NB AND x_2 is PB THEN $y = c_5$
IF x_1 is NS AND x_2 is NB THEN $y = c_6$
IF x_1 is NS AND x_2 is NS THEN $y = c_7$
IF x_1 is NS AND x_2 is Z THEN $y = c_8$
.....
IF x_1 is PB AND x_2 is PS THEN $y = c_{24}$
IF x_1 is PB AND x_2 is PB THEN $y = c_{25}$

Memory address	Memory size	Content
\$0000	1 kB	Even MFs (input x_1)
\$0400	1 kB	Odd MFs (input x_1)
\$0800	1 kB	Index array (input x_1)
\$0C00	1 kB	Even MFs (input x_2)
\$1000	1 kB	Odd MFs (input x_2)
\$1400	1 kB	Index array (input x_2)
\$1800	4 kB	Rule base

Fig. 4. Memory allocation for two-input controller

All data about the membership functions of all FLC input variables are quantified in advance and stored in separate memory blocks as shown in Fig. 4. The proper construction of these blocks is a prerequisite for implementing the faster algorithm of the FLC. Here it is assumed that the input variables x_1 and x_2 are measured by 10-bit ADCs and the FLC output is transferred to an 8-bit DAC. These values are chosen only to illustrate the approach, and other types of ADCs and DACs will change only the size of the blocks, while a different number of inputs will change the number of blocks in Fig. 4.

At each sample instant the DSP executes the following steps:

1. Initialize pointers to the starting addresses of all memory blocks as per Fig. 4.
2. Read the output from the ADC – one value for x_1 and one for x_2 that are used further as offset values.
3. Fuzzification - retrieve the degrees of membership from memory locations whose addresses are obtained by adding the offsets from Step 2 to the pointers from Step 1. Example:

The reading from x_1 is \$267. The memory locations with the degrees of membership for x_1 are:

for even MFs : \$267 + \$0000 = \$0267
for the odd MFs : \$267 + \$0400 = \$0667
for index array : \$267 + \$0800 = \$0A67

4. Detect the fired (activated) fuzzy sets for the measured inputs - retrieve the respective indices from memory location whose address is obtained by adding the offsets from Step 2 to the pointers from Step 1. Example:

The reading from input1 is \$267. The indices of fired fuzzy sets are located at address \$267 + \$0800 = \$0A67, where the index of the active even MF is in the higher 4 bits and the index of the active odd MF in the lower 4 bits.

Memory location	Index of the active even MF	Index of the active odd MF
\$0A67	0010	0001

Obviously, the odd index can be extracted by masking the content with a mask \$0F and the even index by right shifting with 4 positions followed by masking with \$0F.

5. Detect the fired rules within the FRHC - combine the indices of all fired fuzzy sets from Step 4 into rule-indices and then retrieve the consequent parts of the fired rules from the rule base memory block. Example:

Active membership functions for x_1 are NB (even, index = 0) and Z (odd, index = 1) and for x_2 : Z (odd, index = 1) and PB (even, index = 2). The fired rules are:

Rule 1. IF x_1 is NB and x_2 is Z THEN output is c_1
Rule 2. IF x_1 is NB and x_2 is PB THEN output is c_2
Rule 3. IF x_1 is Z and x_2 is Z THEN output is c_3
Rule 4. IF x_1 is Z and x_2 is PB THEN output is c_1

To find the consequent parts of these 4 fired rules we use again the indices of the fired fuzzy sets. Using the pointer to the rule base memory the relative address of each rule in the rule base is formed by combining the indices of the fired fuzzy sets of each input. For the example this gives {NB, Z}, {NB, PB}, {Z, Z} and {Z, PB}. Retrieving from the index arrays for x_1 and x_2 , for the example the content is:

x_1 - NB (index=0) and Z (index=1) => index array location contains \$01 = 0000|0001
 x_2 - Z (index=1) and PB (index=2) => index array location contains \$12 = 0001|0010

From these four 4-bit pieces, the relative and absolute addresses of each of the fired four rules are obtained:

Rule No.	Relative address	Absolute address
1	0000 0001 = \$01	\$1800 + \$01 = \$1801
2	0000 0010 = \$02	\$1800 + \$02 = \$1802
3	0001 0001 = \$11	\$1800 + \$11 = \$1811
4	0001 0010 = \$12	\$1800 + \$12 = \$1812

Reading directly from the absolute addresses we take the singleton values of the consequent parts of the fired rules.

6. Obtain the firing strength of each rule by applying one of the following operators (the mostly used t -norms):

- **min**-operator - $\min(\text{degree_of_membership}_1, \text{degree_of_membership}_2)$;
- **product**-operator - $(\text{degree_of_membership}_1 \times \text{degree_of_membership}_2)$, where “ \times ” means multiplication.

7. Defuzzification using the standard formula:

$$y = \frac{\sum_{i=1}^4 \alpha_i c_i}{\sum_{i=1}^4 \alpha_i}$$

where α_i is the rule firing strength and c_i is the rule consequent part, i is rule number. The output of the controller y has to be 8-bit (just for simplicity). In practice, this formula is computed using the DSP-specific operations for sum of products and division and the result is sent to the controlled process by an appropriate DAC.

Development of the fast fuzzy logic controller

The development cycle consists of three steps. First, the FLC is designed and tuned using MATLAB with Simulink and the Fuzzy Logic Toolbox, as shown in Fig. 5. This allows the development of all necessary preparatory operations such as modeling, simulation, parameter tuning and optimization in the same environment. The resulting FLC is saved in a file in a MATLAB-specific format for later use or archiving. In the second step, the data that describe the FLC completely in terms of memory blocks are prepared, as discussed in Fig. 3 and 4. The universes of discourse of the input variables are quantified in order to obtain the degree of membership of each point of the input space for each fuzzy set. In addition, each point is associated with an index value corresponding to the membership function active at that point. Then the rule base is coded using the same index system in such a way that the index of active membership functions lead to the corresponding parts of the rule base. At the final step, a binary file containing the quantified membership functions, index arrays and the rule base is created out of the stored MATLAB-file. This binary file is then downloaded into the DSP system memory or can be used for burning a programmable ROM to be inserted into the DSP system. Of course, for complete functioning of the DSP-based FLC, the described FRHC-algorithm has to be also downloaded to the DSP memory.

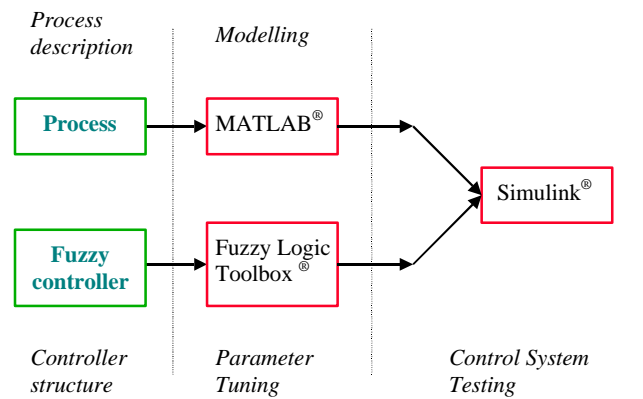


Fig. 5. MATLAB design of fuzzy logic control system

Implementation example

The described structure and principle of operation has been implemented on several DSP platforms. To achieve low cost for the entire FLC, the corresponding low cost starter development kits (SDK) with TMS320C6211 and TMS320C64xx (simulated) were used. However, the standard analog input/output circuits on the SDKs are designed for audio signals and cannot provide enough speed for analog-to-digital conversion. Therefore, the SDKs were upgraded by the addition of fast ADC and DAC circuits which also allow processing signals down to DC. The software is programmed in C in Code Composer Studio™, including Compile Tools and Debug Tools. The resulting assembly code is optimized for speed. A comparison of the implementations is provided in Table 2 together with some technical data of the SDKs used. For an additional comparison, the same algorithm is implemented on the SDK of the popular RISC-type microcontroller Atmel MEGA with integrated ADC.

Table 2. Comparison of FLC on various DSP platforms

DSP platform	Clock frequency	Processor cycles	Sampling interval/frequency	Performance acceleration
TMS320C64xx	600 MHz	13	22 nsec / 45 MHz	3600
TMS320C6211	150 MHz	13	87 nsec / 11 MHz	880
Atmel MEGA 103L	4 MHz	320	80 μsec / 12,5 kHz	1

All cases are tested by a simple illustrative FLC. In fact, the complexity of the FLC is irrelevant to the performance of the DSP implementation, as the FRHC-based algorithm requires a constant number of operations

at each sampling interval. The results in Table 2 show only a small difference in the number of DSP cycles to execute the FLC. This is because the FRHC algorithm requires only fixed point operations that are executed in a similar way on all the DSPs used in the experiments. The difference is bigger in sampling interval/frequency as the clock frequencies of the SDKs are not the same. Significantly, the particular properties of the FRHC-algorithm have no need for a floating point DSP to implement this kind of FLC.

Conclusion

The proposed approach to implementing the FRHC concept on DSP has several advantages. Firstly, instead of executing fuzzification based on a parametric description of the membership functions and searching the fired rules from the total rule-base, the FLC employs only memory-read operations. Secondly, the defuzzification is done quickly, as special properties of the DSP architecture are utilized. Thirdly, the hardware needed to implement this fast FLC is simple and the popular SDKs provide good performance/price output.

The high performance of this approach serves as a basis for further applications, for example, in image- and vision-based control systems where DSPs are already used for some typical processing algorithms. The FRHC-based algorithm is appropriate for implementing certain fuzzy logic based image processing algorithms such as edge detection and filtering, as currently carried out in the authors' research labs.

References

- Del Campo, I., J. Echanobe and J. Tarela (1998) Implementation of intelligent controllers on digital signal processors, *Cybernetics and Systems*, vol.29, no.3, p.283-301.
- Del Campo, I. and J. Tarela (1995) A CAD tool to implement real-time fuzzy controllers on DSPs, *Proc. of 7th Euromicro Workshop on Real-Time Systems*, IEEE Comp. Soc. Press, Los Alamitos, CA, USA, p.323-329.
- George, M. (1996) DSPs make fuzzy logic practical for industrial control, *I&CS*, vol.69, no. 2, p.49-58.
- George, M. (1992) Fuzzy logic servo motor control on the Texas Instruments TMS320C14 DSP, *2nd Intern. Workshop on Ind. Fuzzy Control and Intel.Systems*. Texas A&M Univ., College Station, TX, USA, p.49-56.
- Kalaykov, I. (1998) Fuzzy controllers can be extremely fast. *Preprints of the Nat. Conf. on Automation and Informatics, Symposium on Intelligent Control*, Sofia, Bulgaria, October, vol. 2, p. 31-36.
- Kalaykov, I. (1999) New speed limits of the fuzzy controller hardware. *Proc. Of the 42nd Midwest Symposium on Circuits and Systems, MWSCAS'99*, Las Cruces, NM, USA, August.
- Kumbla, K., T. Akbarzadeh and M. Jamshidi (1995) TMS320 DSP based neuro-fuzzy controller, *IEEE Intern. Conf. on Systems, Man and Cybernetics "Intel. Systems for 21 Century"*, New York, USA, vol. 5, p.4015-20.

Acknowledgements

This work was performed in the Center for Applied Autonomous Sensor Systems at Örebro University, Sweden, and forms part of the research programs sponsored by the KK-foundation, whose support is gratefully acknowledged. Part of the work was done at the Department of Electrical and Computer Engineering, University of New Brunswick, Fredericton, Canada, whose contribution is also acknowledged.

Authors profile

Dr. Ivan Kalaykov is an Associate Professor and Head of the Intelligent Control Lab in the Center of Applied Autonomous Sensor Systems (AASS) at Örebro University, Sweden. His research interests include fuzzy logic and vision based control, fuzzy hardware for fast processes. He has implemented a super fast fuzzy controller on FPGA able to operate at 50 MHz sampling frequency. His experience covers computer controlled systems, embedded systems and DSP applications for various industrial applications.

Mr. Boyko Iliev is a Ph.D. student in the Intelligent Control Lab in the Center of Applied Autonomous Sensor Systems (AASS) at Örebro University, Sweden. His research interests include fuzzy logic control, sliding mode control for robot control and other industrial applications.

Dr. Richard Tervo is a Professor in the Department of Electrical and Computer Engineering at the University of New Brunswick, Fredericton, Canada. He was recently a visiting professor in the Intelligent Control Lab in the Center of Applied Autonomous Sensor Systems (AASS) at Örebro University, Sweden. His research interests include communications, signal processing and DSP applications.