AN FPGA IMPLEMENTATION OF A SELF-TUNED FUZZY CONTROLLER

Kam-Wing Li,
Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont., Canada M5S 1A4.
kli@fuzzy.ie.utoronto.ca

I. Burhan Türkşen , and
Department of Industrial Engineering, University of Toronto, Toronto, Ont., Canada, M5S 1A4.

Kenneth C. Smith
Departments of Electrical and Computer Engineering, Mechanical Engineering, Computer Science, Information Science, University of Toronto, Toronto, Ont., Canada M5S 1A4, and The University of Science and Technology, Hong Kong.

Abstract

This paper presents a field programmable gate array (FPGA) implementation of a self-tuned fuzzy controller which integrates a previous FPGA controller design [1] with a self-tuning mechanism on the same FPGA device.

1. Introduction

There are a variety of different ways of implementing a fuzzy inference engine. The software-oriented approach in which the inference engine is coded in a software program that runs on a general-purpose computer is flexible but lacks speed [2]. The hardware-oriented approach in which the inference engine is mapped onto dedicated hardware allows much faster operation but lacks flexibility [3]. As well, there are compromise approaches in which some dedicated units are incorporated into an otherwise general-purpose computer architecture [4] to allow higher-speed operation while at the same time retaining much of the flexibility of a general-purpose computer approach. The advent of high density field-programmable gate arrays (FPGA) makes it possible to design dedicated-hardware fuzzy-inference engines relatively easily, and with a high degree of flexibility.

Typical rule-based fuzzy controllers often require some amount of tuning, and there are various techniques to automate this tuning process. In this work, the simulated annealing (SA) technique, which is a probabilistic search approach, is employed as the self-tuning mechanism for the membership functions associated with the fuzzy rules. This paper presents an FPGA implementation of a self-tuned fuzzy controller which integrates an earlier FPGA design of a fuzzy controller [1] with an SA-tuning mechanism, on the same FPGA device.

2. A self-tuned fuzzy controller

The block diagram of a closed-loop SA-tuned fuzzy control arrangement is shown in Fig. 1. Essentially, there are two major parts in the SA-tuned fuzzy controller shown :

(i) A fuzzy logic controller

(ii) An SA-tuning mechanism

The fuzzy controller carries out inference operation at high-speed, whereas the tuning procedure works at a much lower rate. For the implementation described in this paper, a two-input-one-output fuzzy controller is considered. Both the inputs and the output have 8-bit resolution, and up to seven membership functions for each input or output can be defined over the universe of discourse. The fuzzy controller has two levels of pipeline which allows overlapping of the arithmetic as well as inference operations [1]. The SA tuning mechanism adjusts the triangular or singleton membership functions to minimize a cost function [5].

The complete self-tuned fuzzy inference engine is implemented in a Xilinx 4000 series FPGA device. This paper describes various aspects of the implementation of the self-tuned system.

0-7803-3225-3-6/96 $5.00 © 1996 IEEE

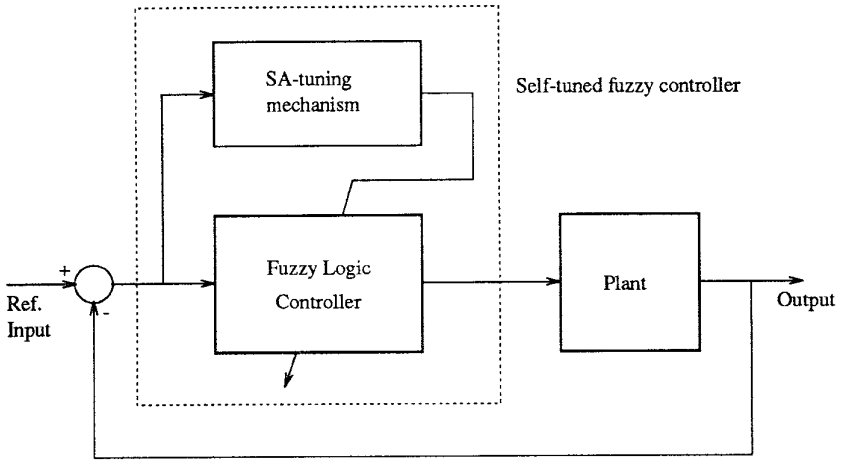

Fig. 1. A closed-loop self-tuned fuzzy control arrangement.

3. The architecture of an SA-tuning mechanism

The SA algorithm used in the self-tuned fuzzy controller can be described briefly as follows:

(i) An antecedent or consequent MF parameter is chosen and perturbed randomly.

(ii) A cost function, $C(\mathbf{w})$, based on the integral of time and absolute error (ITAE) is used to indicate the performance of the system, that is,

$$C(\mathbf{w}) = \int_0^{T_L} t\ |e(t)|\ dt \qquad (1)$$

where $\mathbf{w}$ is the parameter vector, T_L is the length of the time interval for evaluating the cost function, t is the time, and $|e(t)|$ is the absolute error between the reference input and the output.

(iii) According to the Metropolis criterion [6], the perturbed parameter is accepted as a new starting point if there is an improvement in performance, otherwise, it is accepted probabilistically if it leads to a degradation in performance. More formally, it is accepted :

(a) when there is an improvement in performance, namely,

$$C(\mathbf{w}') \leq C(\mathbf{w}) \qquad (2)$$

or,

(b) when there is a deterioration in performance, with a probability of

$$p = e^{\frac{C(w)-C(w')}{T}}, \qquad (3)$$

where $\mathbf{w}'$ is the perturbed parameter vector, p is the probability of acceptance of the perturbed value, and T is a control parameter called the "temperature"

(iv) The process is started with a large value of T, which is reduced by a factor of 0.85 each time when the process reaches a "thermal equilibrium". Here, this is assumed to occur after the number of trials is equal to 5 times the total number of MF parameters [5]

(v) A near-optimal configuration of MFs results as $T \rightarrow 0$

The above procedure is carried out after each complete fuzzy control operation, and, as a result the speed of the tuning mechanism is not paramount. The mapping of the SA-algorithm to hardware is based on a general-purpose hardware architecture to reduce the number of logic gates required. A simplified block diagram of the datapath of the SA portion of the design is shown in Fig. 2. As can be seen, this is quite similar to some general-purpose microprocessor designs using dedicated functional blocks for various parameters.

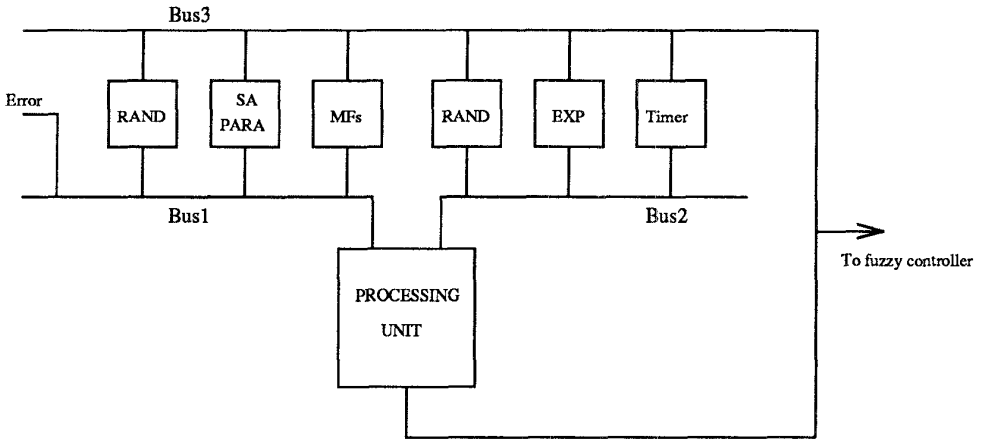

Fig. 2. The datapath for the SA-tuning mechanism.

The processing unit in the Figure carries out addition, subtraction, division, and multiplication, with the latter two based on shift-and-add, or shift-and-subtract operations that require 8 clock cycles to complete. The value of the timer is used as one of the operands in the cost function (ITAE) calculation, with the input (the error) forming the other operand. To simply the computation requirement, the exponential function required in Eqn. (3) is provided by a 16-byte look-up table (EXP). The MF parameters and the corresponding perturbed values are stored in the MFs block which also provides storage for the set of the best MF parameters attained so far. This set of best MF parameters is used at the beginning of a new temperature value T. Altogether, two 16-byte RAMs are required for the MFs, as shown in Fig. 3. After each perturbation of a parameter, the perturbed value is transferred to the fuzzy controller through Bus 3 labelled in Fig. 2. Other parameters for the SA procedure are stored in the 16-byte RAM parameter block (SA PARA). Two pseudo-random-number generators (RAND), which are constructed with linear feedback shift registers, are used to determine the amount of parameter perturbation, and for the Metropolis acceptance criterion.

The datapath shown in Fig. 2 is controlled by a control unit with a 64-word microprogram. Approximately 3,000 gates are needed to implement the SA-tuning mechanism containing both the datapath and the control unit. Another 4,000 gates are required by the fuzzy controller [1]. The 7,000 gates required for the complete SA-tuned fuzzy controller have been implemented on a Xilinx XC4010 FPGA whose maximum capacity is about 10,000 gates. The system runs on a 24 MHz clock.

During normal fuzzy-control operation, the input (error) value and the timer value are multiplied and accumulated continuously every nine clock cycles to form the cost function. At the end of a complete fuzzy-control operation, the cost value is used in the SA procedure described above to determine whether the current MF configuration is acceptable or not. A new perturbed MF-parameter vector is then prepared for the next fuzzy-control operation, and the process is repeated.

The evaluation of the Metropolis criterion requires between 23 to 40 clock cycles, whereas the parameter perturbation requires 45 clock cycles. Therefore, a typical SA procedure requires between 70 to 90 clock cycles to complete. An additional 30 cycles are required for temperature scheduling.

Conclusions

An FPGA implementation of an SA-tuned fuzzy controller has been described. The total number of gates required for the controller is about

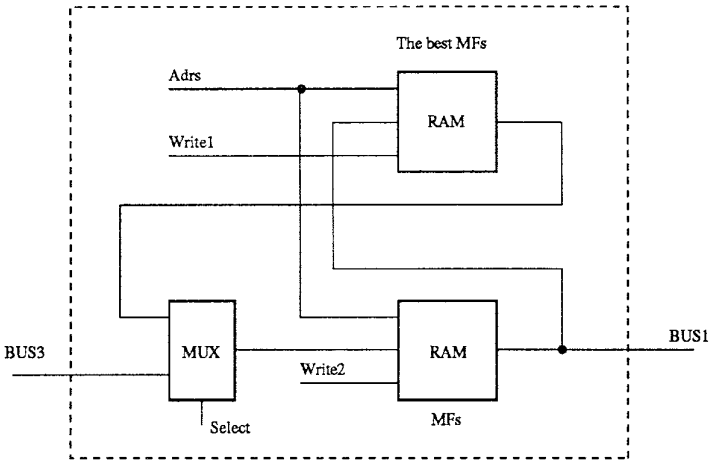

Fig. 3. The internal organization of the MFs block.

7,000, and the controller has been implemented on a Xilinx XC4010 FPGA. The evaluation of the SA algorithm requires about 100 clock cycles.

References

[1] K.-W. Li, I. B. Turksen, and K. C. Smith, "An FPGA implementation of a fuzzy controller based on the product-sum inference method," *Proc. 2nd Joint Conf. on Information Sciences*, Wrightsville Beach, N. C., pp. 80-83, Sept. 1995.

[2] Y. Saito and T. Ishida, "A high speed software fuzzy inference controller," *Proc. 3rd IFSA World Congress*, Aug. 1989, pp. 12-15.

[3] K. Nakamura, et al., "A 12-bit resolution 200 kflips fuzzy inference processor," *IEICE Trans. Electron.*, vol. E76-C, no. 7, pp. 1102-1111, July 1993.

[4] A. P. Ungering, H. Bauer, and K. Goser, "Architecture of a fuzzy-processor based on an 8-bit microprocessor," *Proc. 3rd IEEE Intl. Conf. on Fuzzy Systems*, pp. 297-301, Orlando, Florida, June 1994.

[5] K.-W. Li, "Two-phase fuzzy control," Ph.D. Dissertation, University of Toronto, April 1996.

[6] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087-1091, June 1953.