International Journal of Fuzzy Logic and Intelligent Systems, vol. 5, no. 3, September 2005 pp. 274-279

Very High-speed Integer Fuzzy Controller Using VHDL

Sang Gu Lee* and John D. Carpinelli**

*** Department of Computer Engineering, Hannam University, Korea**
**** Department of Electrical and Computer Engineering, NJIT, USA**

Abstract

For high-speed fuzzy control systems, an important problem is the improvement of speed for the fuzzy inference, particularly in the consequent part and the defuzzification stage. This paper introduces an algorithm to map real values of the fuzzy membership functions in the consequent part onto an integer grid, as well as a method of eliminating the unnecessary operations of the zero items in the defuzzification stage, allowing a center of gravity method to be implemented with only integer additions and one integer division. A VHDL implementation of the system is presented. The proposed system shows approximately an order of magnitude increase in speed as compared with conventional methods while introducing only a minimal error and can be used in many fuzzy controller applications.

Key Words : Fuzzy hardware, VHDL, Integer operation, Defuzzification, COG

1. Introduction

Fuzzy logic models are successfully being used in many applications, including medical diagnosis, data mining, and pattern recognition [7]. Most conventional fuzzy controllers use floating-point operations in [0,1] for fuzzy computing [2,4] or a lookup table (LUT) method [6] for computing membership functions.

In fuzzy sets, all properties are expressed using membership functions of the sets involved and the union, intersection, and complements of these sets. Many floating-point operations are required to calculate output values.

Defuzzification is a mapping from a space of fuzzy control actions defined over an output universe of discourse into a space of crisp control actions. For this operation, again a large number of floating-point operations are performed. Thus, defuzzification is usually one of the most time-consuming procedures in fuzzy processing. Most fuzzy systems use the center of gravity (COG) method to perform defuzzification, as does the present paper.

In general arithmetic operations, the difference in the execution speed between integer operations and floating-point operations is quite large, typically a factor of ten. Although this speed is different for each CPU, integer multiplication and division are about ten times faster than floating point multiplication and division. In cases involving complex and large volumes of computations, as found in typical fuzzy systems, the difference of speed can dominate overall system performance.

Manuscript received Jul. 11, 2005; revised Sep. 7, 2005
This work was supported by the research grant of the Hannam University in 2004

To overcome these problems, this paper uses Bresenham's scan line algorithm [3] to map the real values to a 400 × 30 integer grid. A method of eliminating the unnecessary operations of the continuous zero items in the defuzzification stage is also proposed. The proposed system is applied to a truck backer-upper control system for performance evaluation by software simulation. The proposed system is an order of magnitude faster than the conventional methods using floating-point operations and introduces only a minimal error.

The rest of this paper is organized as follows. Section 2 describes the proposed algorithm for the integer operations in the consequent part and the defuzzification stage. We present the integer mapping algorithm for the consequent part, the data structure for the defuzzification stage, and the proposed COG operations using integer addition operations. Section 3 describes the structure of the fuzzy processor developed to implement the proposed algorithms. The following section describes the simulation and analyzes the performance of the algorithm using both high-level software implementation and VHDL synthesis. Finally, concluding remarks are presented.

2. Very High-speed Fuzzy System Using Integer Operations

In fuzzy inferencing, if fuzzy inputs are given as singleton values, it takes little times to find α, the degree of fulfillment, in each fuzzy rule. However, in the consequent part, it takes much more time due to the many real-valued operations in [0, 1] corresponding to the universe of discourse. Also, in the defuzzification stage, in order to find the center of gravity (COG), many floating-point operations are required.

In this paper, we propose a new method to use only integer operations in the consequent part and the defuzzification stage. Here, we use an integer pixel grid in the consequent part composed of 400 × 30 pixels. After computing the α value, it is multiplied by 30 and converted to an integer, β, by rounding. The β value is transferred to the consequent as a modified degree of fulfillment.

2.1 Integer Mapping Algorithm

Integer mapping of the fuzzy membership function is the first step in the proposed algorithm. We use Bresenham's algorithm [3] to calculate quantized y values in the given line using only integer addition operations. Removing floating point operations produces significant improvement in speed as compared to conventional methods. In the integer grid, a line can be represented by connecting the integer pixels from the starting point pixel to the ending point pixel.

Using a mid-point technique [1], it is very efficient to compute the next y pixel value at given x pixel value using only integer additions. This procedure is as follows.

Let $F(x,y) = ax + by + c = 0$. If $dy = y_1 - y_0$, and $dx = x_1 - x_0$, then $y = \frac{dy}{dx}x + B$, where B is the y-intercept, (x_0,y_0) is the start point, and (x_1,y_1) is the end point.

$\therefore F(x,y) = dy \cdot x - dx \cdot y + Bdx = 0$, where $a = dy$, $b = -dx$, and $c = Bdx$.

To choose the next pixel, the first midpoint is $(x_p+1, y_p+\frac{1}{2})$. We need only to compute the value of $F(x_p+1, y_p+\frac{1}{2})$, and to test its sign. If the sign is positive, we select point $(x + 1, y + 1)$ as a next pixel. If the sign is negative, we select the point $(x + 1, y)$. If d is the decision variable for this choice, then

$$d_{start} = F(x_p+1, y_p+\frac{1}{2}) = a(x_p+1) + b(y_p+\frac{1}{2}) + c$$

$$= (ax_p + by_p + c) + a + \frac{b}{2} = a + \frac{b}{2} \rightarrow 2dy - dx$$

If $d < 0$, then the next pixel is $(x_p + 1, y_p)$, so we select 'E'(East).

If $d \geq 0$, then the next pixel is $(x_p + 1, y_p + 1)$, so we select 'NE' (North East).

If 'E' is selected, the next mid-point is incremented by 1 in the x direction. To find the next pixel, we have to compute the amount of variation, ΔE, and update the decision variable d.

$$d_{new} = F(x_p+2, y_p+\frac{1}{2}) = a(x_p+2) + b(y_p+\frac{1}{2}) + c$$

$$= 2a + \frac{b}{2} \rightarrow 4dy - dx$$

$$d_{old} = F(x_p+1, y_p+\frac{1}{2}) = a(x_p+1) + b(y_p+\frac{1}{2}) + c$$

$$= a + \frac{b}{2} \rightarrow 2dy - dx$$

$$\Delta E = d_{new} - d_{old} = 2dy$$

If 'NE' is selected, then $\Delta NE = d_{new} - d_{old} = 2(dy - dx)$.

Following the above procedures, we test the sign of the decision variable d, and update ΔE or ΔNE, depending on the choice of the next pixel, incrementing x by 1 in every step.■

All of the pixels can be found using integer operations in this procedure. Figure 1 shows the algorithm for integer line mapping with start point (x_1,y_1) and end point (x_2,y_2). Here, β is the modified degree of fulfillment and has an integer value in the range $0 \leq \beta \leq 30$. The value defuzz(xa) is the value of the membership function in the consequent part when the value of the integer pixel in the x-axis is xa. Figure 2 shows the construction of a line segment from point (5,8) to (9,11) using this algorithm.

```
procedure
left_line
begin
  dx←x2-x1
  dy←y2-y1
  d←2dy-dx
  xa←x1
  ya←y1
  a←β
  defuzz(xa)←ya
  while ya < a+1 do
    begin
      xa←xa+1
      begin
       if (d<0)
         d←d+2dy
       else
         ya←ya+1
         d←d+2(dy-dx)
      end
      defuzz(xa)←ya
    end
end.
```

Figure 1. Integer mapping algorithm (left_line).

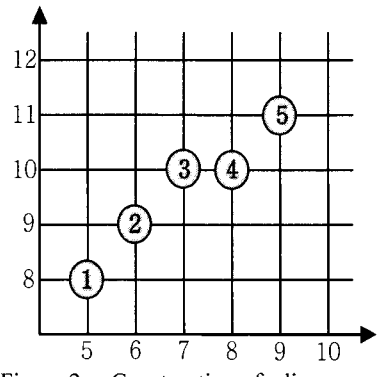

Figure 2. Construction of a line segment.

Next, we describe the mapping procedures of the line (b) and (c), the remaining portions of the membership function, as shown in Figure 3.

Because the required part in the defuzzification stage is in trapezoidal form, the part from integer β+1 to 30 in the y-axis point does not need mapping. Therefore, we must have mappings until the value of y is β. In order to map the membership function in Figure 3, we first process line segment

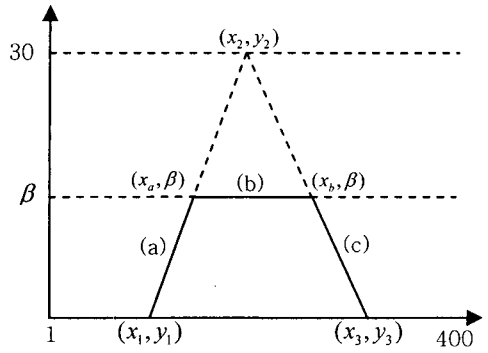

Figure 3. Representation of the consequent part using β

(a), then (c), and finally (b), following the order (a)→(c)→(b). Following this order calculates the endpoints of segment (b) before it is processed.

The process for calculating the integer mapping in line (c) is similar to the process used to map line (a), except x is decremented for each iteration.

$$d_{start} = F(x_p - 1, y_p + \frac{1}{2}) = a(x_p - 1) + b(y_p + \frac{1}{2}) + c = -a + \frac{b}{2}$$
$$\rightarrow -2dy - dx$$

If $d < 0$, we select 'NW'. If $d \geq 0$, we select 'W'.

If 'NW' is selected, the next mid-point is decremented by 1 in the x direction and is incremented by 1 in the y direction. To find the next pixel, we compute the amount of variation, ΔNW, and update the decision variable d.

$\Delta NW = d_{new} - d_{old} = -2dy - 2dx = -2(dy + dx)$ If 'W' is selected, as the similar way, $\Delta W = d_{new} - d_{old} = -2dy$

After mapping lines (a) and (c) to the integer pixels, the final task is the mapping of line (b) to the grid. To map line (b), we must set $y = \beta$ for all pixels from point (x_a, β) to (x_b, β). These two points are the ending points previously computed in lines (a) and (c). Therefore, all y values are set to β in the interval of the x-axis from x_a to x_b.

This integer mapping method offers a significant advantage over the methods that use floating-point operations, as it avoids the floating-point operations. However, this does require a tradeoff. In this algorithm, a quantization error between the real y value and its equivalent integer pixel y value exists because this method selects the nearer integer pixel among two neighboring pixels. This quantization error is inversely proportional to the number of y-axis integer pixels. The root mean square value of the quantization error decreases as the number of y-axis integer pixels increases [5]. In spite of this drawback, the increase in speed justifies the use of this procedure for application that can tolerate the small error it introduces. In practice, the size of the y-axis pixels can be adjusted according to the application domain.

2.2 Data Structure in the Defuzzification

In this paper, we use COG method for defuzzification. The required data structure in the defuzzification stage is an integer array that can store the y-axis integer values corresponding to the 400 x-axis integer pixels in the consequent part. In this array, defuzz(x), the y-axis integer values from 0 to 30 are stored. On computing the consequent part, this array must contain the maximum values in all fuzzy rules. Figure 4 shows the algorithm for this function.

```
begin
  integer array defuzz[1:m]
  integer array max[1:m]←0

  for i := 1 step 1 until n do
  begin
    for j := 1 step 1 until m do
    Begin
      operation of integer pixel mapping
      return value ← defuzz(1:m)
    if max(j) < defuzz(j)
      then max(j) ← defuzz (j)
    end
  end
end.
```

Figure 4. Update algorithm of Max value

Here, n is the number of the fuzzy rules, and m is the number of the pixels in the x-axis.

After computing the consequent part, in general, many values of *defuzz(x)* have values of zero continuously from either end of the array. It is not necessary to compute any operations in that interval less than *lower* or greater than *upper* in the defuzzification process. To overcome this problem, we check x_1 and x_3 for each fuzzy rule, setting *lower* to the minimum of all x_1 and *upper* to the maximum of all x_3.

2.3 New COG Operation Without Multiplications

In conventional fuzzy systems, the center of gravity, computation requires $2(n\text{-}1)$ additions, n multiplications and one division. In this paper, we also propose an algorithm to reduce the computation times for the multiplications that uses only integer additions to calculate the nominator of the COG operation. One integer division operation is still needed to compute the final COG. Figure 5 shows the algorithm to compute the nominator of the COG function.

```
for i := upper step = −1 until lower
  begin
```

```
    temp ← defuzz(i) + temp
    sum ← sum + temp
  end
COG ← sum / temp
COG ← COG + (lower − 1)
```

Figure 5. The proposed algorithm for COG operations

The final "temp" value in Figure 5 becomes the denominator. In this method, for n non-zero items, only $2n$ additions and 1 division are required. Table 1 compares the conventional method without considering the non-zero items, the conventional method considering the non-zero items, and the proposed method. Note that the proposed method does not require any multiplications. Here, $1 \le lower \le upper \le 400$.

Table 1: Comparison of arithmetic operations in Center of Gravity calculations

	additions	multiplications	divisions
Conventional method without considering non-zero items	2 × (400 − 1)	400	1
Conventional method considering non-zero items	2 × T	T	1
Proposed method	2 × T	0	1

T = upper − lower + 1

3. Hardware Structure of the Proposed Fuzzy Processor

3.1 Entire Structure and Operations

The structure of the proposed one-chip fuzzy processor is shown in Figure 6. It contains modules of rule control interface, integer mapping, defuzzification, and memory. The ROM stores all conditions to be processed from the conditional part in the rule interface.

The operation of the fuzzy processor is activated from the reset signal. First, in the rule control module, the points of (x1, 0), (x2, 30), (x3, 0) and β in the ROM are transferred to the mapping module. In the mapping module, y-coordinates (0-30) are computed according to the x-coordinates (1-400) and this module compares that value and the content of the addressed RAM so that the larger one is saved to RAM.

If start_rule = '1', then the computed y value is saved to RAM, and all '0' data is saved to the addresses (1−x1) and (x3−400). After saving, this module signals to the rule control module in order to receive the next data. After receiving the final data, the control signal is sent to the Defuzzy module. After reading the contents in the RAM, this module performs additions and one division operation and computes and output the COG.

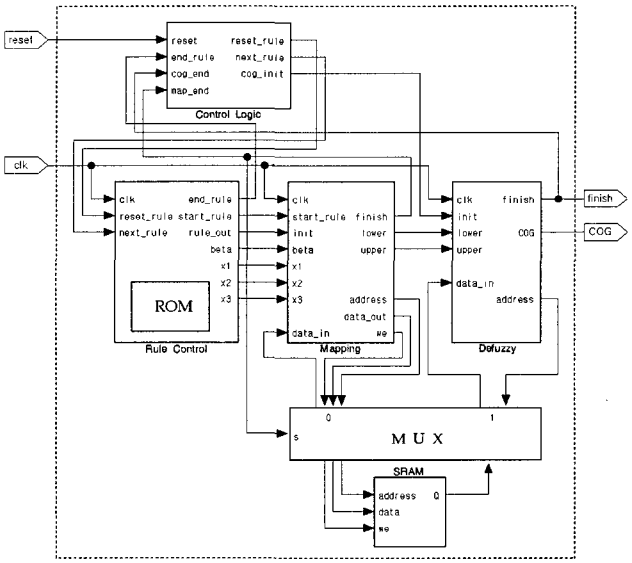

Figure 6. Block diagram of proposed processor

3.2 Rule Control Module

The data format for the rule control module is shown in Figure 7 to process in the conclusion part by using data in conditional part. This data format is composed of 9-bit × 4.

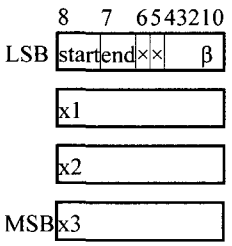

Figure 7. Data format for the rule control module

Here, start_rule and end_rule are control signals for controlling each module. The values of x1, x2, x3 and β are transferred to the mapping module for computing in the conclusion part.

If end_rule = 1, the mapping module is finished, all final results are saved to RAM and control is transferred to the Defuzzy module for defuzzification.

3.3 Mapping Module

The mapping module is composed of Pre_process, Left, Right, Middle, and Zero modules. In mapping module, the following three operations are needed.

(1) With the data from the rule control module, integer mapping processes are performed at each lower module using only integer operations.
(2) The mapping results are compared with RAM and saved to RAM.
(3) The values of lower and upper are updated for the defuzzification process.

To share RAM with the other modules, the finish signal is introduced. So while the mapping module is using memory, the finish signal is reset. The finish signal is set when that module is finished. The structure of the mapping module is shown in Figure 8.

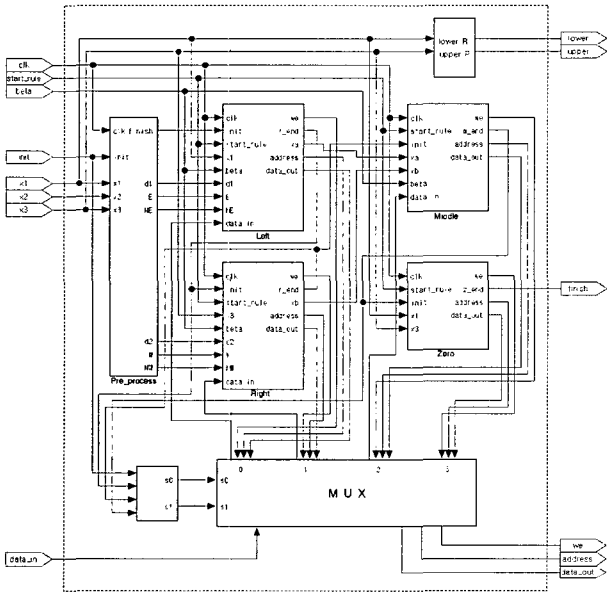

Figure 8. Block diagram of the Mapping module

3.3.1 Pre_process Module

This module inputs the values of x1, x2 and x3 and outputs the values of d1, E, NE, d2, W and NW for initial values of integer mapping operations.

3.3.2 Left Module

This module receives the values of x1, β, d1, N and NE for mapping the (a) part of Figure 3. This module computes the y-coordinate based on the x-coordinate from x1. When rule_start = '0', the greater of the computed value and the stored value in the RAM is saved to RAM; if rule_start = '1', the computed value is saved to RAM. Here, if the computed y value is equal to the β, and the corresponding x value is xa, the value of xa is output and control is transferred to the Right module.

3.3.3 Right Module

This module receives the values of x3, beta, d2, W and NW for mapping the (c) part of Figure 3 in a process similar to that used by the left module. Here, if the computed y value is equal to the β, the corresponding x value of xb is output and control is transferred to Middle module.

3.3.4 Middle module

This module receives the values of xa (from Left), xb (from Right) and beta for mapping the (b) part of Figure 3. This module computes the y-coordinate for x-coordinates from xa. When rule_start = '0', the larger of the value in the RAM and β is saved to RAM. In the case of rule_start = '1', β is saved to RAM. After repeating the same operation until x is equal to xb, control is transferred to the Zero module.

3.3.5 Zero module

When start_rule = '1', it initializes the contents of RAM, setting locations 1-x1 and x3-400 to 0, and outputting the finish signal. When start_rule = '0', no operations are performed and it outputs the finish signal.

3.4 Defuzzy module

In the Defuzzy module, the COG is computed by referencing RAM where the final data in the conclusion part are saved. The defuzzy module is composed of Cog_add and Cog_divide modules; its structure is shown in Figure 9.

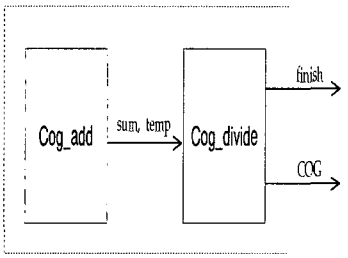

Figure 9. Block diagram of the Defuzzy module

3.5 Memory

To store the pixels, we use 5-bit×512 RAM. Through the MUX, input and output operations to read and write RAM can be performed. We also use 9-bit×512 ROM to interface the data in the condition part. As 9-bit×4 of storage space is needed to input one condition, up to 128 conditions are allowed. We used the LPM_ROM component of LPM Library and LPM_RAM_DQ component for implementing ROM and RAM

4. Simulation and Performance Analysis

To analyze the performance of the proposed method, we simulated it using both high-level programs and VHDL synthesis. First, algorithms of the truck backer-upper control system for both methods were coded in C/C++ and execution

speeds were timed. By using only integer operations, the proposed algorithm executed approximately 12.75 times faster than the traditional algorithm that uses floating point operations.

We also developed a VHDL implementation of the proposed algorithm and compared its performance to that of the traditional algorithm. For this implementation, the proposed algorithm reduced the time needed to calculate system outputs by an order of magnitude. Both the software and VHDL simulation results are consistent with the speedup achieved; see Figure 10.

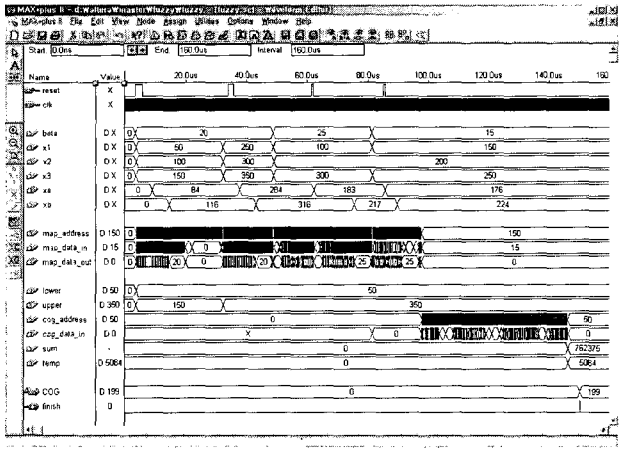

Figure 10. Timing analysis of the VHDL design

Finally, we note that it is possible to improve the performance of the proposed algorithm by utilizing parallel processing techniques. For example, the functions for line segments (a) and (c) in the Figure 3 are independent and can be calculated in parallel. In addition, the points on line segment (b) in that figure are also independent of each other and do not have to be calculated sequentially. For membership functions that are symmetric, it is also possible to calculate only the points on line segment (a) and then make use of symmetry to generate the points on (c) without significant computations.

5. Conclusion

This paper has presented a method to improve the speed of fuzzy control systems by using integer representations of membership functions and integer operations. A method for calculating the COG function without multiplications reduces the execution time of the algorithm. A VHDL implementation is presented. Based on simulation results, the proposed method offers a speedup over the conventional method of approximately an order of magnitude. This system can also be applied to build powerful architectures for control applications, such as robotic control, with time-critical sensor integration.

References

[1] E. S. Angel, *Interactive Computer Graphics, 3rd ed.*, Addison-Wesley, Boston, MA, USA, 2002.

[2] I. Batroune, A. Barriga, Sanchez-Solano, G. J. Jimenez-Fernandez, and D. R. Lopez, *Microelectronic Design of Fuzzy Logic-Based Systems*, CRC Press, New York, 2000.

[3] J. E. Bresenham, "Incremental line compaction," The Computer Journal (1982) 116-120.

[4] A. M. Ibrahim, *Fuzzy Logic for Embedded Systems Applications*, Elsevier, 2004.

[5] F. Marcelloni and M. Aksit, "Fuzzy logic-based object-oriented methods to reduce quantization error and contextual bias problems in software development," *Fuzzy Sets and Systems* 145 (2004) 57-80.

[6] Z. Salcic, "High-speed Customizable Fuzzy-Logic Processor: Architecture and Implementation," *IEEE Trans. Systems, Man, & Cybernetics – Part A* 31 (6) (2001) 731-737.

[7] J. Yen and R. Langari, *Fuzzy Logic: Intelligence,* Prentice Hall, Englewood Cliffs, NJ, USA, 1999.

Sang Gu Lee

He received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1978, and the M.S. degree in Computer Science from KAIST, Korea in 1981. He received the Ph.D. degree in Electrical Electronics and Computer Engineering from Waseda University, Tokyo, Japan. Since 1983, he has been a professor in division of Computer Engineering, Hannam University, Taejon, Korea. His current research interests are in parallel processing, parallel architecture and parallel neuro-fuzzy computing.

Phone : +82-42-629-7551

E-mail: sglee@hannam.ac.kr

John D. Carpinelli

He has been a professor in the Dept. of Electrical and Computer Engineering, New Jersey Institute of Technology, New Jersey, USA. His current research interests are in computer architecture, parallel architecture, interconnection networks, educational computing, fuzzy computing and biomedical computing.

E-mail: carpinelli@njit.edu