Intelligent Control on the S12 Microcontroller Using Fuzzy Logic Instructions

Christopher R. Carroll
University of Minnesota Duluth

Introduction

Intelligent Control is a modern phrase that implies using creative algorithms in computer control applications to address problems in unusual, or "intelligent," ways. One tool that is used to implement Intelligent Control is Fuzzy Logic, a scheme by which computer applications can make decisions on imprecise, incomplete, or "fuzzy" information. This approach to Intelligent Control has seen application in various commercial products, from home appliances to sophisticated system designs. The value of using Fuzzy Logic in such applications depends on the situation. Using Fuzzy Logic to detect and control the "darkness" of a piece of toast in a toaster seems to be a force-fit application, but in more complex situations, Fuzzy Logic allows implementation of non-linear control without complicated mathematical support.

The Freescale S12 microcontroller includes specific instructions in its instruction set to support Fuzzy Logic applications. The presence of these four instructions as primitive operations in the S12 makes that microcontroller unique, and especially well-suited to Intelligent Control applications. This paper details those instructions in the S12's instruction set that implement Fuzzy Logic operations, and provides some applications in which the S12's Fuzzy Logic capabilities are used.

During Spring semester, 2010, a Design Workshop course was offered in which students used the S12 microcontroller to implement applications of Intelligent Control. Based on the experience of teaching that workshop, a similar Design Workshop course is scheduled for Fall semester, 2010. This paper will include some results from the design projects conducted during the Spring workshop as examples of Intelligent Control applications using Fuzzy Logic.

The Freescale S12 processor is probably the most popular general-purpose 16-bit microcontroller currently on the market. It is used as the focus for microprocessor/microcontroller courses in many Electrical or Computer Engineering programs across the country. However, the four instructions in the instruction set that implement Fuzzy Logic primitives are often omitted from discussion because their use requires an understanding that exceeds normal microcontroller applications. This paper tries to remove the mystery surrounding the Fuzzy Logic capabilities of the S12 microcontroller, and demonstrates how they can be used for Intelligent Control.

Overview of the Fuzzy Logic Instructions

The S12 microcontroller includes four primitive instructions in its instruction set specifically intended to support Fuzzy Logic operations. These are MEM, REV, REVW, and WAV, and they are introduced briefly below. Following sections of this paper will discuss the instructions in more detail.

The MEM (membership) instruction performs the first step in Fuzzy Logic operations known as fuzzification of the external crisp input values. This produces a set of fuzzy input variables that are later combined to produce fuzzy output values.

The REV and REVW (rule evaluation and weighted rule evaluation) instructions perform the meat of the fuzzy calculations, using the fuzzy input variables produced by MEM and generating fuzzy output values.

The WAV (weighted average) instruction performs the final step of Fuzzy Logic operations known as defuzzification. It takes the fuzzy output variables and transforms them into crisp system outputs that can then be used in traditional processing.

These three steps, fuzzification, rule evaluation, and defuzzification, form a brief outline of any Fuzzy Logic application, and each step is implemented by a primitive instruction in the S12 microcontroller's instruction set.

MEM Instruction

The MEM (membership) instruction takes crisp input values received from transducers or other devices and generates fuzzy input variables that represent the extent to which the crisp input values belong to certain fuzzy categories, or labels. The crisp input can be fully a member of a certain category, partially a member of that category, or not a member at all. To be specific, imagine a system that controls the water temperature in a shower for people of various ages.

Labels are implemented via trapezoidal membership functions, as depicted in Figure 1. This figure shows five labels for water temperature, each with an associated trapezoidal membership function, to take an input water temperature from a sensor and categorize the temperature as cold, cool, warm, hot, or scalding. It also shows a second set of trapezoidal membership functions that depict three labels, young, adult, and senior, based on the age of the shower user. In each case, the crisp input, (water temperature or age) is represented by a one-byte number in the range 0 to 255 on the horizontal axis, and the resulting fuzzy values are also represented by one-byte numbers in the range 0 to 255 on the vertical axis.

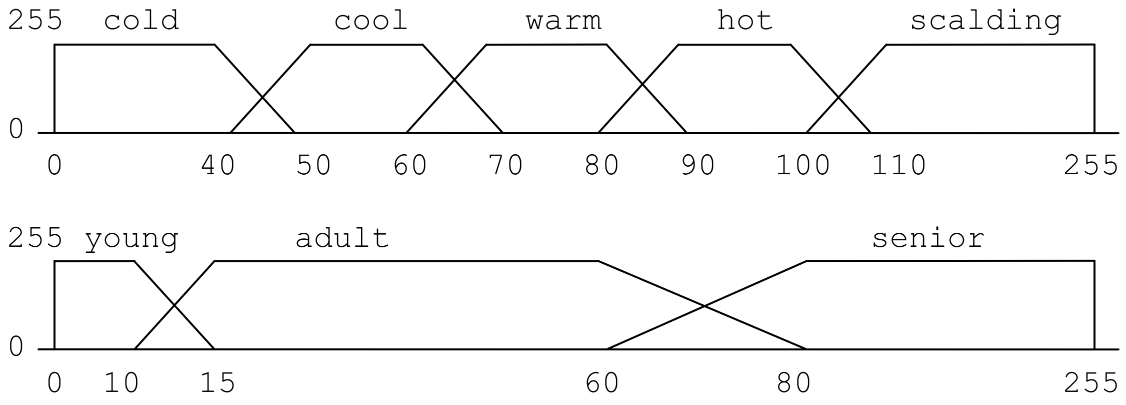

Figure 1. Membership functions that fuzzify two crisp input values, water temperature and age, and produce eight fuzzy input values, cold, cool, warm, hot, scalding, young, adult, and senior.

To use the MEM instruction, the trapezoidal membership functions must be described in the microcontroller. This is accomplished via a data structure that records four numbers for each trapezoid: left x-axis intercept, right x-axis intercept, left slope, and right slope (negated). Thus, the "cool" trapezoid in Figure 1 would be described with the data structure 40, 70, 25, 25 in the S12's memory. Infinite slopes are represented by a special-case slope value of 0. The four values for each trapezoid are stored sequentially in memory.

Before executing MEM, three registers in the S12 CPU must be initialized. Register A holds the crisp value being fuzzified. Register X holds the address of the first byte of the trapezoidal function being evaluated. Register Y holds the address in memory where the resulting fuzzy input value is to be stored. MEM is executed once for each trapezoid in the set of membership functions. In this example, eight separated MEM instructions must be executed, five with register A holding the crisp water temperature, and three with register A holding the crisp age of the shower user.

The result of executing MEM for each of the trapezoids is a list in memory of fuzzy input values representing the extent to which each crisp input is a member of the associated fuzzy category identified by each trapezoid. These fuzzy input values are then used in the next step of the processing.

REV and REVW Instructions

The REV and REVW (rule evaluation and weighted rule evaluation) implement the meat of the Fuzzy Logic processing. Through these instructions, fuzzy inputs produced by MEM are combined using a set of rules to produce fuzzy outputs. The rules are a collection of statements that describe the fuzzy output based on characteristics of the fuzzy inputs. In the example discussed here, the single fuzzy output describes how to change the shower water temperature, given the fuzzy inputs that describe the characteristics of the current measured temperature and the age of the shower user.

Figure 2 shows a typical set of rules that might be used in this shower temperature control example. In the table, the action required to adjust the water temperature is identified for each of the possible categories of water temperature and age of the shower user. Entries in the table mean the following: ↑↑ = raise temperature quickly, ↑ = raise temperature slowly, ↔ = leave temperature unchanged, ↓ = lower temperature slowly, and ↓↓ = lower temperature quickly.

	cold	cool	warm	hot	scalding
young	↑↑	↑	↔	↓	↓↓
adult	↑↑	↑↑	↑	↔	↓↓
senior	↑↑	↑	↔	↓↓	↓↓

Figure 2. Rules used by REV and REVW to generate fuzzy outputs from fuzzy inputs

To use the REV and REVW instructions, the list of rules shown in Figure 2 must be stored in the S12 memory in another data structure. Each box in Figure 2 is represented with a list of bytes that record the statement "If the water temperature is (…) AND the showerer is (…) THEN adjust the water temperature in this way (…)" where each of the (…) represents a fuzzy input produced by MEM or a fuzzy output generated by the REV or REVW instructions. Thus, a sample rule would be "If the water temperature is cool AND the showerer is young THEN raise the temperature slowly."

The difference between REV and REVW is that REV allocates the same "weight" to each rule, meaning that each rule has equal impact on the resulting fuzzy output. By contrast, REVW allows the programmer to assign weights to the various rules so that some rules have more impact on the fuzzy output result than others. The data structures in memory for REV and REVW differ in order to accommodate this weighting feature. In order to avoid confusion, suffice it to say that the data structure specifying the list of rules identifies, for each rule, the fuzzy inputs that are combined with the AND operator, and identifies the fuzzy output that is produced by that rule. Rules are stored consecutively in memory, and are separated by special "marker" values stored between the rules in the data structure. Results of the rules are combined with the OR operator to determine the final value of the fuzzy outputs. In this example, there are five fuzzy outputs: raise the temperature quickly, raise the temperature slowly, leave the temperature unchanged, lower the temperature slowly, and lower the temperature quickly.

Numerically, the AND operator in Fuzzy Logic is implemented as an arithmetic minimum operator, so that the AND of two fuzzy inputs is just the minimum value of the two fuzzy values. The OR operator in Fuzzy Logic is implemented as an arithmetic maximum operator, so that when rule results are combined by ORing them, the fuzzy output value is just the maximum value produced by each of the rules being combined.

The result of rule evaluation is a set of fuzzy output values indicating the extent to which each of the output actions should be taken. Thus, in this example, five numbers are produced in the range 0 to 255, one for each of the five actions that should be taken on adjusting the water temperature. This form of the result is not particularly useful for the system that actually must control the water temperature. Thus, there is one more step in the process.

WAV Instruction

The WAV (weighted average) instruction takes the fuzzy outputs produced by REV or REVW and combines them to produce a crisp output value that can then be used in further traditional processing. This is accomplished by assigning a set of ideal values, known as "singletons," to each of the fuzzy outputs, and then performing a weighted average calculation using the fuzzy outputs as "weights" to condition the singleton values associated with each fuzzy output. This calculation is much the same as a "center of mass" calculation in a mechanical system. The resulting number is a value somewhere within the limits established by the specified singleton values, determined by the fuzzy outputs that specify the extent to which the output should represent each of the fuzzy output labels.

In this example, if the fuzzy outputs say that the water temperature should be raised slowly to a large degree, left unchanged to a small degree, and lowered not at all, the resulting value after WAV will be a value between the singleton values for raise slowly and leave unchanged, shaded toward the raise slowly singleton according to the weights identified in the fuzzy output variables.

Example Applications

Students in the Electrical and Computer Engineering Design Workshop course during Spring semester, 2010, used the S10 microcontroller and its Fuzzy Logic instructions to implement various applications of Intelligent Control. Some of these student projects are described here.

In the "Intelligent Greenhouse" project, students designed a system that controls the environment of growing plants. The system measures temperature and humidity in a greenhouse atmosphere and uses those values as crisp inputs to the system. Employing Fuzzy Logic, the system generates signals to control heat and ventilation of the greenhouse to optimize conditions for plant growth. The results of this project were hard to demonstrate, but plants did grow, so something must have been right.

In the "Color Recognition for Tracking Robots" project, students designed a typical line-following robot, but added a twist. The color of the line being tracked controlled the speed of the robot. A green line indicated full speed. A blue line slowed the robot, and a red line caused the robot to stop. Filtered sensors were used to detect the color of the line, and Fuzzy Logic was used to combine the crisp sensor outputs and to generate the control signal to specify robot speed. This project worked well, after some difficulty in properly sensing the different colors.

The "Path Tracking" project was also based on a line-following robot, but in this case the line sometimes included alternate paths which could be selected by the robot, based on the surrounding environment and the intended destination. The robot used infrared sensors to detect the line, and ultrasonic sensors to detect surrounding obstacles. By combining the line-tracking sensor information with information about surrounding obstacles, the robot was able to make intelligent decisions when faced with a bifurcation in the path it was following.

Two projects attempted to use spoken commands to control a system. One project included a multi-color light display, and the user could light one or more colors of lights by speaking the color. Colors could be brightened or darkened by speaking "more" or "less" as well. This project did not function well. The second voice-control project did better, using spoken commands to control the speed and direction of a motor. The S12 processor does not have any special support for signal processing, so these projects attempted to just capture the frequency pattern of spoken input and analyzed that pattern to determine appropriate actions.

A final project equipped a motorcycle helmet with ultrasonic sensors to detect surrounding obstacles. A "threat level" indication was provided for the helmet wearer to indicate the presence and direction of detected obstacles. Intelligent Control attempted to analyze the threat situation and report the severity of the threat via lights in the peripheral vision of the user.

Student reactions to using Fuzzy Logic to implement control systems for their projects ranged widely. Some students appreciated the opportunity to implement a control scheme using a state-of-the-art technique, and eagerly dove into their projects. Other students were not convinced that the use of Fuzzy Logic in their projects justified the added complexity in their software required to support that approach. It is true that with the level of complexity addressed here in these student projects, the full benefit available through the power of Fuzzy Logic systems is not realized. In more complicated cases, however, Fuzzy Logic can be used effectively to implement cleanly a control system that otherwise would require many levels of mathematical modeling and simulation.

Summary

Intelligent Control applications were successfully implemented by students in the Design Workshop Class during Spring semester, 2010, using the Fuzzy Logic capabilities of the S12 microcontroller. No conclusion is drawn here that Fuzzy Logic is the best, or even an appropriate vehicle for solving these problems, but the availability of Fuzzy Logic support instructions in the S12's instruction set makes the approach at least viable. Experience with these primitive applications of intelligent control using Fuzzy Logic demonstrated the processing power that is available through special features of today's systems. That experience may encourage instructors in microprocessor classes using the S12 processor to address the capabilities available through the Fuzzy Logic instructions in its instruction set.

References

1. Carroll, C. R., and M. Stachowicz, "Fuzzy Logic on the MC68HC12 Microcontroller: A Student Design Workshop," *Computers in Education Journal*, Vol XI, No. 1, January-March (2001).

2. Stachowicz, M. and C. R. Carroll, "Fuzzy Logic on Motorola's Microcontroller," *3rd Working Conference on Engineering Education: Engineering Education for the 21st Century*, Sheffield Hallam University, England (2000).

3. Stachowicz, M. and C. Carroll, "Intelligent Systems on Motorola's Microcontroller: A Team Design Workshop," *Proceedings of the ICEE-2000*, Taipei, Taiwan (2000).

4. Carroll, C. R., and M. Stachowicz, "Fuzzy Logic on the MC68HC12 Microcontroller: A Student Design Workshop," *2000 ASEE Annual Conference Proceedings*, St. Louis, MO (2000).

Biography

CHRISTOPHER R. CARROLL received a Bachelor degree from Georgia Tech, and M.S. and Ph.D. degrees from Caltech. After teaching at Duke University, he is now Associate Professor and Assistant Head of Electrical and Computer Engineering at UMD, with interests in special-purpose digital systems, VLSI, and microprocessors.