

FUZZY LOGIC KERNEL USING 68HC11

T u t o r i a l

© 1998

GUNAWAN SETIABUDI

FUZZY LOGIC KERNEL USING M68HC11

By. Gunawan Setiabudi ©1998

This is the first edition of fuzzy logic kernel for micro controller 68HC11 using the C language as the programming language. All of the program here has been tested and guaranteed to run. To compile the fuzzy logic kernel I use the XGCCHC11 compiler by Oliver Kraus (oliver@cip.e-technik.uni-erlangen.de).

OVERVIEW

Fuzzy logic is new kind of control system. It was found by Prof. Lotfi Zadeh in 1965. Word “fuzzy” means something blur, not clear, but please don’t be mistaken, because the result of its control system is very reliable and very stable, specially on a nonlinear control system. In fuzzy logic the operators only need to input the linguistic input, output and rules without any complex calculation or maths. So what fuzzy logic needs more is the experienced operator not a genius in math to calculate a silly transfer function that is needed in a conventional control system.

The M68HC11 is an 8-bit microcontroller from Motorola Inc. This microcontroller has a build-in EEPROM, and in 68HC11A1FN there are 512 bytes EEPROM. I need EEPROM to store the knowledge base that send from the knowledge base generator. And this microcontroller also has 4 channels build-in 8-bit ADC, and also has 2 8-bit paralel outputs that can be easily connected to the DAC to get the analog output.

I write the fuzzy logic kernel to improve the 68HC11 capabilities to control a system better and easier. But because this kernel is written in C language, it also can be used with another microcontroller with a little bit modification. The capabilities of

this fuzzy logic kernel is: able to receive 4 inputs, control 2 outputs and evaluate up to 30 rules. It use the port E0 – E3 as the analog input and port B as the first analog output and port C as the second analog output.

Fuzzy Logic

Fuzzy Logic is a kind of logic that has been extended to understand not only completely true or completely false (like the Boolean Logic did) but also understand the degree between 0 to 1. Or according Prof. Lotfi Zadeh, the definition of Fuzzy Logic is “Computing With Words.”

In fuzzy logic, the user do not need to deal with complex calculation, but only deal with the combination of rules, such as “If Temperature is Hot Then Fan is Fast.” This simple rule has 2 parts, that is:

- ⇒ The IF part. This part also called as *Antecedent*. This is the input’s condition. In the previous example, the input is temperature and the condition is hot and the phrase “temperature is hot” is the antecedent.
- ⇒ The THEN part, or also called *Consequent*. This is the output’s condition. In the example, the output is Fan and the condition is Fast and the phrase “Fan is Fast” is the consequent.

In fuzzy logic, the condition is called *Membership Function*. So for the input temperature, the membership function is hot. Ussualy each input has more than 1 membership, the most common number of membership function is an odd number from 3 to 7. Why is it ussualy an odd number number? Because if you use the odd number, so input or output can be drawn simetrically.

The advantage of using rules and linguistic descriptions, you can represent a condition with a degree. For example, if you define the 20°C to 24°C is a warm

condition, then what about $19,9^{\circ}\text{C}$? In Boolean logic it isn't a warm condition anymore, but in fuzzy logic, it is nearly warm and the 20°C is completely warm. So the changing for each condition happens very smooth.

The other advantage is if use fuzzy logic in a non-linear system. This logic can control a non-linear system very well because it uses the rules and the linguistic description which is very flexible. For example, fuzzy logic has been used in the harbour to control the smooth movement of a container transportation from the ship to the harbour.

Now what about the disadvantage using a fuzzy logic? If you use the fuzzy logic in a linear system, it's still a good alternative but it's quite expensive if you compare it with the conventional logic that use analog circuit.

The other disadvantage is, the word "fuzzy." Since it means something that not clear, for some people who has mis-understanding that could be a problem. "How can I control something using a method that is not clear?" perhaps this is the question for that kind of person. This reason made fuzzy logic not so popular in USA but it's grown very fast in Japan, where most people didn't care about the word "fuzzy."

To use the fuzzy logic in control system, you have to follow this 3 steps: fuzzification, Rule Evaluation, and Defuzzification. Figure 1 is the description for the steps in fuzzy logic.

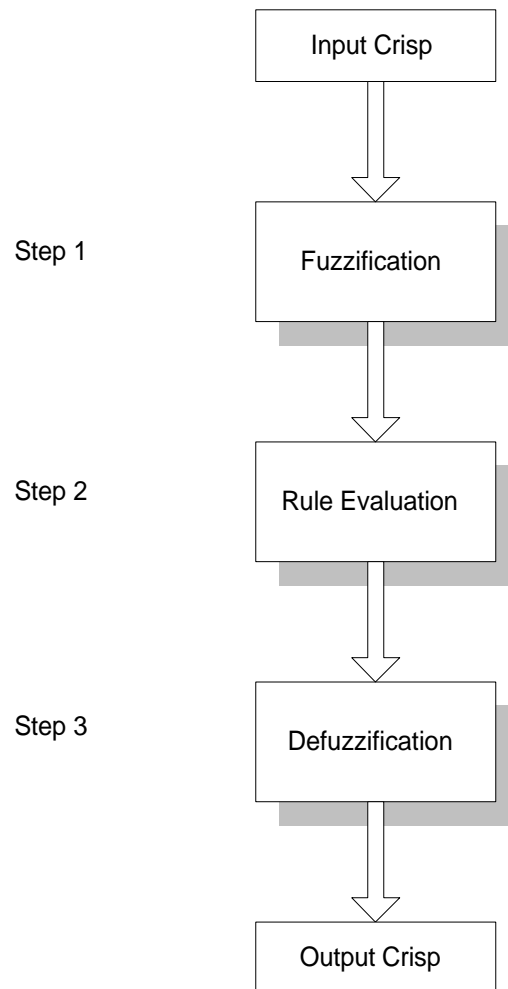


Figure 1
Steps in Fuzzy Logic

Crisp input is the “real-world” input and usually it is an input from the sensors, and it’s usually an analog value.

The first step in fuzzy logic is fuzzification. This step will convert the crisp input as a fuzzy input. For an example: the temperature 20°C will be converted as warm. From this example, the 20°C is the crisp input and the warm is the fuzzy input. Because the membership function usually intersect with another membership

function, so it is possible if one condition has 2 membership functions. Figure 2 is the example for an input that has 2 membership functions.

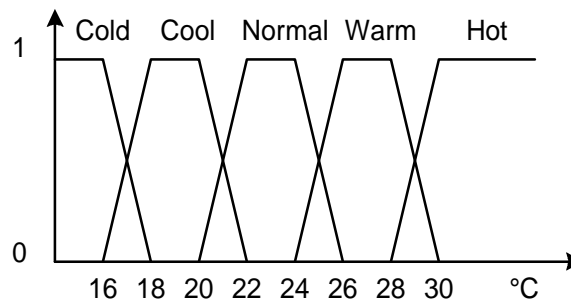


Figure 2

Example of Fuzzification Process

From this example you can see that 16.5°C has 2 membership functions that are 0.25 cool and 0.75 cold. It makes every changing for every membership function can happen smoothly.

The second step is rule evaluation. This step will evaluate each rule from the knowledge base and the result is the appropriate condition. After the fuzzification step, each input will be compared with the rules and then the result of the rule evaluation is the most correct conditions. In my fuzzy logic kernel I use the Min-Max Method. This method will looking for the minimum value of the antecedent for the each rule to find the rule strength and then this value will be compared with the other rule strength and then the overall maximum value will be fuzzy output.

And the last step is defuzzification. This process is the inversion of fuzzification process, that is convert the fuzzy output to crisp output. The fuzzy output is the result from rule evaluation and crisp output is the value that will be used in the system. In fuzzy logic kernel I use the Centre-Of-Gravity Method (COG

Method). I choose this method this method only needs a simple calculation and it can simpler and faster if you use the singleton membership function. For the brief explanation about singleton membership function you can find it in the data structure section. Using the COG method I can calculate the crisp output using the formula below:

$$\frac{\sum_{i=1}^n Fi * Si}{\sum_{i=1}^n Fi}$$

Where:

F is Fuzzy Output from the rule evaluation process.

S is Singleton from the knowledge base.

N is the number of singleton.

After the crisp output calculated then this output is the final output from your system to the application.

68HC11 Micro Controller

HC11 is an 8-bit micro controller made by Motorola Inc. If you want to ask some information about HC11 or ask some literature, you can reach them through their web site that has been linked in my home page.

This micro controller has a lot of build-in peripherals but I only use some of these peripherals such as, EEPROM, ADC and timer for the delay. I use the ADC to capture the inputs, and the EEPROM to store the knowledge base. Why do I use EEPROM to store the knowledge base? Because by using the EEPROM, I can store the data and the data will not change even when I shut down the power and I

am also able to write the data into it by using the program and I don't have to add any circuits at all.

If you want to program or erase the EEPROM, then you have to deal with the PPROG register. The figure below is the figure of PPROG register.

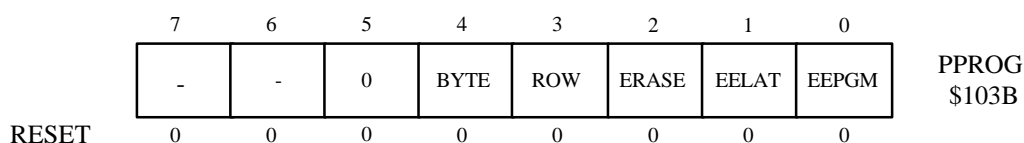


Figure 3
PPROG Register

For the explanation for each bit, you can read the HC11 reference manual. This is the algorithm to write any data to any EEPROM's address.

1. Set EELAT bit in PPROG register, so EEPROM will be configured for programming or erasing process.
2. Write data at desired address.
3. Set EEGPM bit in PPROG register, so Vpp power activated.
4. Give necessary delay. If you use E-clock with frequency 2 MHz, then the delay needed is 10 ms.
5. Reset PPROG register.

To write or to erase a data, you need a delay sub-routine. For the delay sub-routine I use the build-in timer in HC11. There are 5 timers, but I only use 1 of them, that is the second timer.

Before you write any data into EEPROM I suggest you to erase the EEPROM first. And this is the algorithm to erase the whole data in EEPROM.

1. Set ERASE and EELAT bit in PPROG register.
2. Write any data at EEPROM's address.
3. Set EEGPM bit in PPROG register to activate the Vpp power.
4. Give necessary delay. If you use E-clock with frequency 2 MHz, then the delay needed is 10 ms.
5. Reset PPROG register.

To capture the inputs, I use the build-in ADC. To use it, I have to capture the input using port E0 to E3, that means the maximum input is limited by the hardware capabilities, and the maximum input is 4. For the output I use the paralel output port B as the first output and port C as the second output. You only able to use the second output if the first output has been used. And for the rules, it should able to evaluate up to 61 rules (assuming there are 4 inputs and 2 outputs), but because the fuzzy logic kernel evaluate the rules sequentially, so to make fuzzy logic kernel works faster and more optimal, I suggest the maximum rules are 30. And This is the algorithm to use the ADC.

1. At the first 64 cycles, set ADPU bit at OPTION register.
2. Set the ADC mode, number of ADC that you want to use and the which ADC you want to use at ADCTL register.
3. Wait until CCF bit at ADCTL register set.
4. The conversion result is stored at ADR1 to ADR4, depends on which ADC you want to use.

I don't discuss about the registers very detail, so I hope you can refer to the 68HC11 reference manual to learn about them.

Data Structure

This section will explain how to store the information about the linguistic data into another data format that can be understood by the fuzzy logic kernel, and the memory mapping in the HC11's EEPROM. To understand how to build a fuzzy logic kernel, you have to understand this section. But if you only want to use the fuzzy logic kernel, you can skip this section.

To represent the membership function, I use 2 types. The first type is trapesoidal type and the second type is the singleton type. I use the first type to

represent the input membership function and the second type to represent the output membership function. Why do I use the trapezoidal type instead of triangular type? The reason of this, is because the trapezoidal type has 2 advantages those are: with using the trapezoidal type it is possible to draw a range of point (Example from 20°C to 24°C is 100% warm) and the second advantage is from the trapezoidal type you also able to draw the triangular type (with combining the top range into 1 value, ex. Only 22°C is 100% warm). But this kind of membership function has 1 disadvantage that is it needs 4 bytes to represent 1 membership function (if you use the triangular type you only need 3 bytes).

To make it easier to understand, assume the figure 3 is the example of trapezoidal membership function and the Y axis is the degree of membership and the X axis is the range of possible input value. In this example, I use the temperature example (again!!) and the full scale is 30°C.

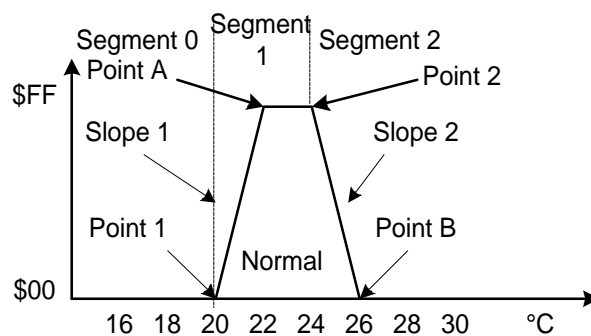


Figure 4

Example of Trapezoidal Membership Function

And to describe this membership function I need 4 bytes.

- ⇒ First byte to store the first point and the range is from \$00 to \$FF. The formula to calculate the first point is: $(\text{input's coordinate}/\text{full scale}) * \FF . On the example, the first byte will store: $(20/30) * \$FF = \AB .
- ⇒ Second byte to store the first slope and the range is also from \$00 to \$FF. The formula for the slope is: $\$FF/(\text{Point A} - \text{Point 1})$. So, the first thing to calculate is the position of point A. By using the same formula, point A = $(22/30) * \$FF = \BC . And we can calculate the first slope is: \$0F.
- ⇒ Third byte to store the second point and the formula is the same with the formula for first point. From the formula we can get the second point value is: \$CD.
- ⇒ Fourth point to store the second slope and by using the same formula with the first slope, we can get the first slope is: \$0F where the second slope is always assume to be negative.

Now, what's a singleton type? Singleton is a special type of fuzzy set that only has 1 element without any masses. Why do I use this type? Because by using this type I can calculate the crisp output easier. About the calculation and how can it make the calculation easier will be explained in the **fuzzy logic kernel** section.

To represent the linguistic rules, I use 2 formats. The first format is to store the information for the antecedents, and the second format is to store the information for the consequent. How to determine if this is the information for the antecedent or consequent? I use the MSB as the flag. If $MSB = 1$ it means this is the consequent information and if the $MSB = 0$, it means it is the antecedent information.

The rest of the bits are used to store the information about which input and which membership function that will be evaluated. The first bit is to store the

information for the input. For the antecedent, there are maximum 4 inputs so I need 2 bits to store this information. Meanwhile for the consequent, there are 2 outputs, so I need only 1 bit to store the information. Next bit are used to store the information for the membership function. Since there are maximum 8 membership functions, so I need 3 bits to store the information. The rest bits are useless. To make you understand easier, please refer to the figure below:

7	6	5	4	3	2	1	0
0	0	0	X	X	A	A	A

(a)

7	6	5	4	3	2	1	0
1	0	0	0	X	A	A	A

(b)

Figure 5

Format for (a) Antecedent (b) Consequent

Knowledge Base Generator

A knowledge base file is a file that store the information about the system you want to build. So in a fuzzy logic knowledge base file, there will be information about the number of input, input membership function, output, output membership function and rules.

To generate a fuzzy logic knowledge base file you have to use the fuzzy logic knowledge base generator program. This program is made from C language. Another way is using the knowledge base file that generated from FUDGE by

Motorola Inc. Or from the FuzzyTech software. The file that is generated from those file are fully compatible with the knowledge base I use.

The only thing you need to build yourself is the downloader program. You can see the downloading process in the **How To Download Knowledge Base File** section. You can build this program as an exercise, if you found any difficulties in making this program, you can contact me at guns@techie.com.

Fuzzy Logic Kernel

As I've explain in the fuzzy logic section, there are 3 steps in fuzzy logic there are: fuzzification, rule evaluation, defuzzification. In this section I'll explain about how to apply those steps in fuzzy logic kernel and which method I use in fuzzy logic kernel.

The first step is **fuzzification**. This step will convert the crisp input into fuzzy input. For example: if input's reading is 20°C the after the fuzzification process, it will be converted into condition "warm." In this step I use the simplest method, so it can make the controlling process done faster. For inputs membership function, I use the trapesoidal type (you can see the figure in figure 4) and for fuzzification process, I divide the membership function into 3 segment there are:

- ⇒ Segment 0: this segment covers the area from point 1 to the left.
- ⇒ Segment 1: this segment covers the area from point 1 to point 2.
- ⇒ Segment 2: this segment covers the area from point 2 to the right.

After the input captured, then the program will determine in which segment is the input. After the segment found, then the fuzzy logic kernel will calculate the fuzzy input using this formula:

- ⇒ If input is in segment 0 then fuzzy input = 0.

⇒ If input is in segment 1 then fuzzy input=(input - point 1)*slope 1 (the maximum if \$FF).

⇒ If input is in segment 2 then fuzzy input=(point 2 - input)*slope 2 (the minimum value is \$00).

In fuzzy logic kernel the result of this step will be stored at **fuz_ins** variable. Since there are 4 inputs and 8 membership function each, so the fuz_ins variable needs 32 byte to store the results. In C language you can do it by declaring the variable as an array.

The second step is **rule evaluation**. This step will evaluate the rule according the input's condition. For example if the input is "Temperature IS warm" then evaluate only rule which contain condition "temperature IS warm" such as "IF temperature IS warm THEN fan IS fast".

To understand this step I suggest you to re-read the data format to store the rules in the fuzzy logic kernel in **Data Structure** section. From the data structure, we can define if this is an antecedent or an consequent by looking at the MSB and the rest of the data is a pointer to the input or output condition. So in processing the data, first the program have to check the MSB then take the appropriate action.

If the data is identified as an antecedent, then get fuzzy input. You can get the proper fuzzy input by adding the value of rule with the fuz_ins variable. If there is a value besides 0 in the fuz_ins variable it means the rule fit the input's condition, so evaluate this rule. If the value is 0, it means the input is not there, so this is a false rule and then proceed next rule.

Since I use the Min-Max Method, for the antecedent I have to find the minimum value and this minimum value will be the rule strength for each rule. After all rules proceed and has its own strength, then compare each rule strength and find

the maximum value. This maximum value will be the fuzzy output. To make it easier to understand let's see this example:

Rule 1 : IF temperature IS normal(0,25) AND day IS night(0,46) THEN fan IS slow. (rule strength 0,25).

Rule 2 : IF temperature IS warm(0,30) AND day IS evening(0,35) THEN fan IS normal. (rule strength 0,30).

Rule 3 : IF temperature IS cold(0,20) AND day IS evening(0,35) THEN fan IS slow. (rule strength 0,20).

Rule 4 : IF temperature IS normal(0,45) AND day IS morning(0,40) THEN fan IS normal. (rule strength 0,40).

In this example the result of overall rule strength (fuzzy output) is: fan speed is 0,25 slow and 0,40 normal.

After the antecedent processing, next the program will process the consequent part. First the program will mask the MSB with 0, so the consequent will be a pointer for the appropriate singleton. And this singleton will be used in the defuzzification process. The result of this process will be stored at **fuz_out** variable. Since there are 2 outputs and 8 membership function each, so the program needs 16 bytes to store the output. And this variable is also declared as an array.

The last step is **defuzzification**. This process will convert the fuzzy output into crisp output. For example: the condition "fan IS normal" will be converted into 300 RPMs. How can it be done? As I've told in the fuzzy logic section, I use the COG method in this step and to calculate the crisp output I use the COG formula. You can see an example of defuzzification process below.

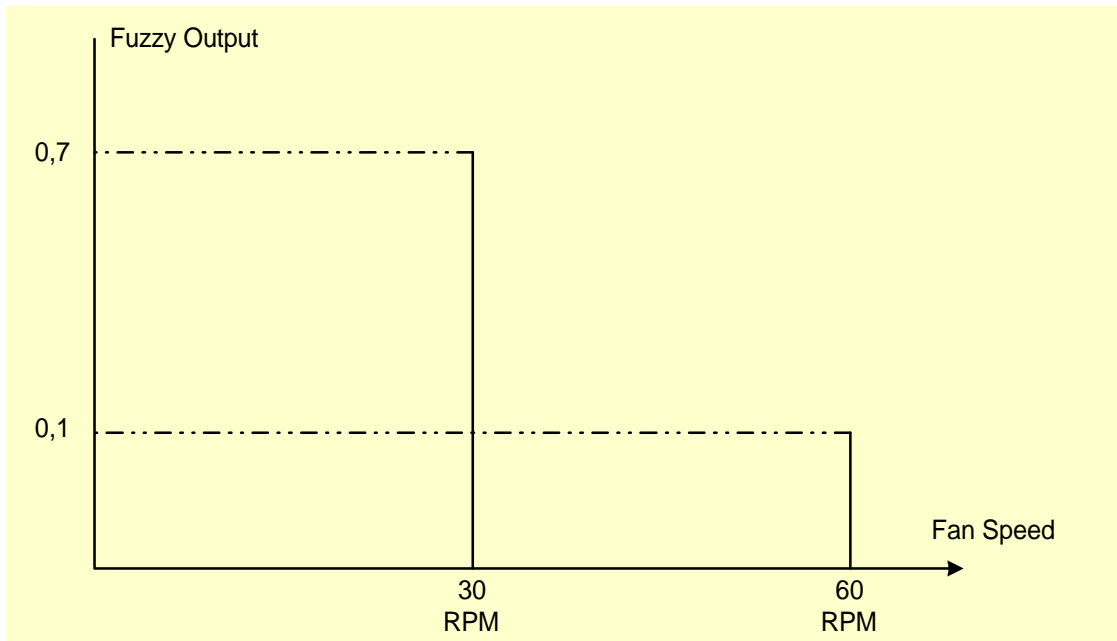


Figure 6
Example of Defuzzification Process

Suppose the figure 6 is the result of rule evaluation process. The fuzzy outputs are 0.7 for singleton 30 RPM and 0.1 for singleton 60 RPM. And to calculate the crisp output we can simply use the COG formula, the result will be:

$$\text{Crisp Output} = \frac{0.7 \times 30 + 0.1 \times 60}{0.7 + 0.1} = 33.75$$

The defuzzification process algorithm is quite simple, but since I use the XGCCHC11 compiler which is not support the long int variable, so I have to use a simple trick. To store the result of singleton and fuzzy output multiplication, I use an array which has 8 elements of integer to store each multiplication. And another variabel to store total of **fuzzy output**. And then a **temp** variabel to store each division result. And the **frac** variable to store the overflow.

If you're using another C compiler that support the long int variable, then you can simplify the source code and get the faster result.

How To Use Fuzzy Logic Kernel

To use the fuzzy logic kernel is very simple, just follow this steps:

- ⇒ If you had downloaded the fuzzy.bin you can skip second step.
- ⇒ Compile the source code using any c compiler. If you use the XGCCHC11 on my site, then go to the tmp directory and run the f.bat followed by the filename. Example: f fuzzy (without the extention). The result of this process is the binary file and you can go to next step.
- ⇒ Download the binary file to your EPROM. If you use the binary file on my site, then use the 8 KB EPROM (type 2764).
- ⇒ Install the EPROM in your circuit.
- ⇒ Connect your inputs to port E0 - port E3. Remember the maximum inputs are 4.
- ⇒ Connect the outputs to port B, if you have 2 outputs, then connect the second output to port C.
- ⇒ Run the knowledge base generator program and build your own knowledge base.
- ⇒ Download the knowledge base to HC11.
- ⇒ Reset the HC11 system.
- ⇒ The fuzzy logic kernel is ready to run.

How To Download Knowledge Base File

To use the fuzzy logic kernel to control your system, first you have to build the knowledge base file and then you have to download it to your HC11 system. This section will discuss about how to download the file and the problems.

To download the file, I use this protocol:

- ⇒ First the HC11 system will send the STX signal (ASCII 02).
- ⇒ The HC11 will wait for about 0.5 ms for the handshake signal from the knowledge base generator program. The expected signal is ETX (ASCII 03).
- ⇒ If the HC11 system receives the ETX then it is a downloading session, if the HC11 system doesn't receive the signal then it's a controlling session.
- ⇒ On downloading session, for every line, the knowledge base generator will send an ETX signal.
- ⇒ On the end of downloading, the knowledge base generator will send the ETB signal. To make it easier to understand, you can see the figure below.

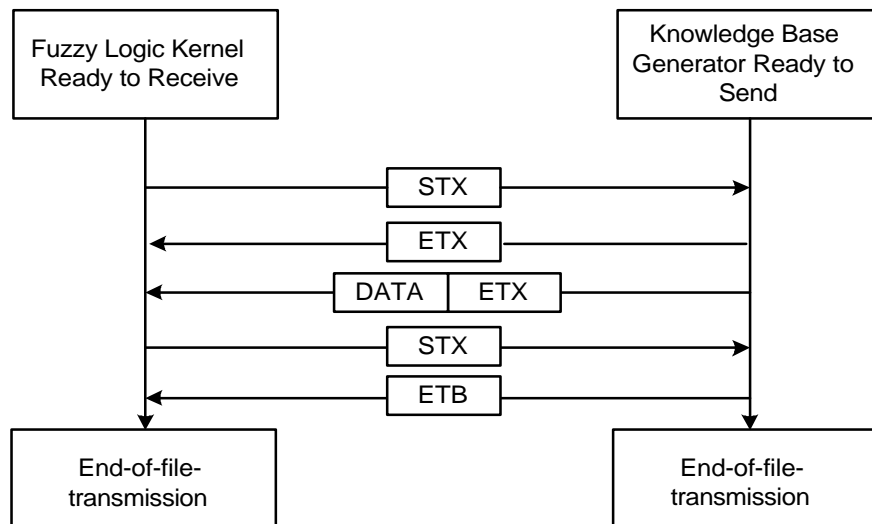


Figure 7

Downloading Process

Because the knowledge base generator program expects the HC11 system to send the signal first, so you have to turn on the fuzzy logic knowledge base program first, then after the waiting message occurs, then reset the HC11 system and the downloading process will be started.

In the downloading process, the most common problem is the “protocol errors.” In this case, you should reset your HC11 system once again, and this would fix the problem. The second problem is the downloading process won’t start at all. To solve this problem, first check your serial cable, if it is okay, then check your comm port is it the correct comm port? And then the next suggestion is check your serial card is it still working? Oh I’m sorry, I forgot the first step, first of all you have to pray, and then you can start to check your problem (this is a plug and pray device) :).

Conclusion

This fuzzy logic kernel is a universal kernel so you can use it to control any system that fulfill its specification. From my personal experience, this kernel can do the controlling job very well, but I haven’t try it on a system that needs a high speed responses. One thing you need to remember, since the rule evaluation process done sequentially, so more rules mean slower response.

Since the source code is written in C language, so you can transport the code to another micro controller that has the C compiler such as PIC, 8051, 8031, etc. To convert the source code, you only need to modify the input source (from PE0-PE3 to the inputs you use), output and EEPROM location.

Nowadays there are 2 kind of fuzzy logic processor. The first kind is a dedicated fuzzy logic processor. This is a processor that specially made to run a

system using fuzzy logic, this kind of processor usually has analog input and analog output. I made the kernel to work like this type of processor. The second type is a general micro controller that support the fuzzy logic. Usually this type of processor has additional instruction sets that support fuzzy logic. To run a system using this type of processor, an operator that capable to write the fuzzy logic kernel is expected. The example of this type of processor is M68HC12 by Motorola Inc.

End-of-tutorial-version-I

If you have any comments, suggestions, questions or projects you want to share to the world please feel free to send them to guns@techie.com . Thanks for visiting my site and hope this tutorial can help you to understand how to apply fuzzy logic on an 8-bit micro controller. If you find out this tutorial is useful please donate USD\$5 to:

Gunawan Setiabudi

Jln. Dr. Sutomo 29B

Telp. 62-0361-232757

Denpasar – Bali

Indonesia