rFLASH User's Guide

Version 1.1
October 2007

Rigel Corporation
P.O. Box 90040
Gainesville, Florida 32607
(352) 384-3766

www.rigelcorp.com

Rigel Corporation's Software License Agreement

This Software License Agreement ("Agreement") covers all software products copyrighted to Rigel Corporation, including but not limited to: Reads51, rLib51, RbHost, RitaBrowser, rFLASH, rrFLASH, rChpSim, Reads166, and rFLI.

This Agreement is between an individual user or a single entity and Rigel Corporation. It applies to all Rigel Corporation software products. These Products ("Products") includes computer software and associated electronic media or documentation "online" or otherwise.

Our software, help files, examples, and related text files may be used without fee by students, faculty and staff of academic institutions and by individuals for non-commercial use. For distribution rights and all other users, including corporate use, please contact:
Rigel Corporation, PO Box 90040, Gainesville, FL 32607
or e-mail techsupport@rigelcorp.com

Terms and Conditions of the Agreement

1. These Products are protected by copyright laws, intellectual property laws, and international treaties. Rigel Corporation owns the title, copyright, and all other intellectual property rights in these Products. We grant you a personal, non-transferable, and non-exclusive license to use the Products. These Products are not transferred to you, given away to you or sold to you.

   Non-commercial use: These Products are licensed to you free of charge.

   Commercial use: You must contact Rigel Corporation to find out if a licensing fee applies before using these Products.

2. You may install and use an unlimited number of copies of these Products.

3. You may store copies of these Products on a storage device or a network for your own use.

4. You may not reproduce and distribute these Products to other parties by electronic means or over computer or communication networks. You may not transfer these Products to a third party. You may not rent, lease, or lend these Products.

5. You may not modify, disassemble, reverse engineer, or translate these Products.

6. These Products are provided by Rigel Corporation "as is" with all faults.

7. In no event shall Rigel Corporation be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the Product, even if Rigel Corporation has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitations of consequential or incidental damages, the above limitations may not apply to you.

8. Rigel Corporation makes no claims as to the applicability or suitability of these Products to your particular use, application, or implementation.

9. Rigel Corporation reserves all rights not expressly granted to you in this Agreement.

10 If you do not abide by or violate the terms and conditions of this Agreement, without prejudice to any other rights, Rigel Corporation may cancel this Agreement. If Rigel Corporation cancels this Agreement; you must remove and destroy all copies of these Products.

11. If you acquired this Product in the United States of America, this Agreement is governed by the laws of the Great State of Florida. If this Product was acquired outside the United States of America all pertinent international treaties apply.

Hardware Warranty

Limited Warranty. Rigel Corporation warrants, for a period of sixty (60) days from your receipt, that READS software, RROS, hardware assembled boards and hardware unassembled components shall be free of substantial errors or defects in material and workmanship which will materially interfere with the proper operation of the items purchased. If you believe such an error or defect exists, please call Rigel Corporation at (352) 373-4629 to see whether such error or defect may be corrected, prior to returning items to Rigel Corporation. Rigel Corporation will repair or replace, at its sole discretion, any defective items, at no cost to you, and the foregoing shall constitute your sole and exclusive remedy in the event of any defects in material or workmanship. Although Rigel Corporation warranty covers 60 days, Rigel shall not be responsible for malfunctions due to customer errors, this includes but is not limited to, errors in connecting the board to power or external circuitry. This warranty does not apply to products which have been subject to misuse (including static discharge), neglect, accident or modification, or which have been soldered or altered during assembly and are not capable of being tested.

DO NOT USE PRODUCTS SOLD BY RIGEL CORPORATION AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS!

Products sold by Rigel Corporation are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

THE LIMITED WARRANTIES SET FORTH HEREIN ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

YOU ASSUME ALL RISKS AND LIABILITY FROM OPERATION OF ITEMS PURCHASED AND RIGEL CORPORATION SHALL IN NO EVENT BE LIABLE FOR DAMAGES CAUSED BY USE OR PERFORMANCE, FOR LOSS PROFITS, PERSONAL INJURY OR FOR ANY OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES. RIGEL CORPORATION'S LIABILITY SHALL NOT EXCEED THE COST OF REPAIR OR REPLACEMENT OF DEFECTIVE ITEMS.

IF THE FOREGOING LIMITATIONS ON LIABILITY ARE UNACCEPTABLE TO YOU, YOU SHOULD RETURN ALL ITEMS PURCHASED TO RIGEL CORPORATION PRIOR TO USE.

Return Policy. This policy applies only when product purchased directly from Rigel Corporation. If you are not satisfied with the items purchased, **prior to usage**, you may return them to Rigel Corporation within thirty (30) days of your receipt of same and receive a full refund from Rigel Corporation. This does not apply to books. Books are non-returnable.

Please call (352) 373-4629 to receive an RMA (Returned Merchandise Authorization) number prior to returning product. You will be responsible for shipping costs.

All returns must be made within 30 days of date of invoice and be accompanied by the original invoice number and a brief explanation of the reason for the return.

Return merchandise in original packaging.

All returned products are subject to a $15 restocking charge. "Custom Items" are not returnable.

Repair Policy. If you encounter problems with your board or software after the 60 day warranty period, please call Rigel Corporation at (352) 384-3766 or email techsupport@rigelcorp.com for advice and instruction.

Rigel Corporation will test and attempt to repair any board. You will be responsible for shipping costs and repair fees. If you send a detailed report of the problems you encountered while operating the board, Rigel Corporation will inspect and test your board to determine what the problem is free of charge. Rigel Corporation will then contact you with an estimated repair bill. You will have the choice of having the board fixed, returned to you as is, or purchasing a new board at a reduced price. Rigel Corporation charges repair fees based on an hourly rate of $50.00. Any parts that need to be replaced will be charged as separate items. Although Rigel Corporation will test and repair any board, it shall not be responsible for malfunctions due to customer errors, this includes but is not limited to, errors in connecting the board to power or external circuitry.

Board Kit. If you are purchasing a board kit, you are assumed to have the skill and knowledge necessary to properly assemble same. Please inspect all components and review accompanying instructions. If instructions are unclear, please return the kit unassembled for a full refund or, if you prefer, Rigel Corporation will send you an assembled and tested board and bill you the price difference. You shall be responsible for shipping costs. The foregoing shall apply only where the kit is unassembled. In the event the kit is partially assembled, a refund will not be available, however, Rigel Corporation can, upon request, complete assembly for a fee based on an hourly rate of $50.00. Although Rigel Corporation will replace any defective parts, it shall not be responsible for malfunctions due to errors in assembly. If you encounter problems with assembly, please call Rigel Corporation at (352) 373-4629 for advice and instruction. In the event a problem cannot be resolved by telephone, Rigel Corporation will perform repair work, upon request, at the foregoing rate of $50.00 per hour.

Governing Law. This agreement and all rights of the respective parties shall be governed by the laws of the State of Florida.

Table of Contents

1 INTRODUCTION

rFLASH (Rigel's Fuzzy Logic Applications Software Helper) is a code generator which creates a set of subroutines and tables in the MCS-51 assembly language to implement Fuzzy Logic Control (FLC) applications. The code generated runs on the 8051 family of microcontrollers. rFLASH v1.10 is a Windows implementation of the DOS version of Rigel Corporation's FLASH software. It runs under all Windows operating systems, Win95 or newer.

1.1 Fuzzy Logic Control

Fuzzy Logic Control (FLC) is based on the theory of Fuzzy Sets, receiving a lot of recent attention (see bibliography). It has become popular with the increasing use of microcontrollers and embedded controllers. FLC can perhaps be best classified as a high-level programming paradigm. It is more suitable for embedded control when digital information processing is used, such as in the case of microcontrollers. A wide range of non-linear control tasks may be implemented with FLC. Since FLC is described by rules written in plain (English) language, no special programming skills are required. This feature makes FLC easy to review and modify. FLC is considered to be a robust methodology since typically the system response does not depend on a single rule or even a single input.

Advantages of FLC

Suitable for microcontroller-based systems.
High level linguistic-variable-based control task description.
Easy to review and modify.
Robust.
May be used for linear and nonlinear control.
Easy to implement multi-input-multi-output control.
Smooth response.

Disadvantages of FLC

Difficult to program
Difficult to model and evaluate with analytical methods.

FLC requires some numerical manipulations which are difficult to program separately for each application. Moreover, unlike, say PID (proportional-integral-differential) control, since FLC is not necessarily a single-input-single-output linear control paradigm, analytical methods to evaluate the controller's response and behavior do not exist. rFLASH attempts to address the disadvantages of FLC. As a code generator, rFLASH creates the FLC code directly from a high-level Control Task Description File (CTDF). rFLASH also has a simulator that will generate the outputs from given inputs on the PC. This simulator may be used to test several inputs and fine-tune the terms or rules accordingly.

1.2 Added Features of rFLASH v1.10

Features that were added to version v1.10 fall into three categories:

1. Internal features that have to do with the operating system memory management and file management;
2. Reads51 version 4 toolchain support for relative assembly and C compiler support;
3. Additional compile options.

1.2.1 Compile Options

The following compile options are added:

Register bank use

rFLASH uses only one register bank. You may specify the register bank to be used by the "bankN" directive, where N is 0 to 3. If the register bank is not specified, the currently selected register bank is used. For example,

```
#define bank0
```

uses the register bank 3. If a register bank is specified, the currently used register bank is saved by pushing PSW on stack. This register bank is re-selected after the iteration by popping PSW.

Word ordering

By default, the word variables (16-bit inputs and outputs) are stored in a big-endian fashion. That is, the high byte of the word is stored in memory with the lower address. Note that most C compilers, including the Reads51 C compiler, use the little-endian ordering. The following two directives are provided:

```
#define little_endian
#define big_endian
```

The default is big_endian to maintain compatibility with the older version of the software.

1.2.2 Reads51 Projects

The examples are implemented as Reads51 projects. The main routines are in C. The assembly language output of rFLASH is included as a module. A function prototype is needed in the calling program to execute the fuzzy-logic routines.

```
// prototype
void rFLASHPoll(void);
```

It is a good idea to keep the rFLASH source file (*.f) in the project in the "exclude from build" group.

Hints

Syntactic and semantic errors are provided by the compiler in the output window. Ranges of input variables for which the outputs are not defined may be observed by enabling the "Tabulate I/O" option. Note that these do not necessarily imply errors. It is possible to reduce the rule set if certain input ranges are known never to occur.

2 BASICS OF FUZZY LOGIC CONTROL WITH rFLASH

The basics of Fuzzy Logic Control (FLC) are discussed in this section. A climate control task consisting of a heating and an air conditioning system is used as an example to illustrate the operational principles of rFLASH. The example is elaborated and extended through this section.

FLC consist of 6 stages as shown below.

Stages					
Crisp Inputs	Fuzzification	Rule Evaluation	Inference and Aggregate	Defuzzification	Crisp Output
Read inputs from internal or external data memory	Compute membership grade of each input for each term	Compute membership grade for each condition of each rule and compute membership grade for each rule	Create a set of output singletons	Compute a single value for each output Alternatives: (1) Centroid (Center-of-Gravity Method) (2) Minimax	Place output into internal or external data memory

The input to FLC consists of the numerical values, such as speed, temperature, or light intensity. Similarly the outputs are values such as motor current or switch position. Although these values are numerical, FLC works with non-numerical linguistic variables. The inputs must first be translated into this linguistic domain. Similarly, after the appropriate response is determined, the response must then be translated from the linguistic domain into the numerical values. These stages are referred to as the fuzzification (numerical input values to linguistic values) and the defuzzification (linguistic values to numerical output values) stages. The core of FLC is the rule base. It consists of the rules, written with the linguistic variables and their terms. The Rule Evaluation and Inferencing stages are strongly coupled. Inferencing refers to scanning the rules and constructing a set of outputs, each with its own weight. Defuzzification may either pick the output with the largest weight, referred to as the minimax defuzzification method, or compute the weighted sum (average) of the set of outputs, referred to as the centroid or center-of-gravity defuzzification method.

Linguistic variables are the names of the input and output attributes. Consider, for example, a climate control task. *Temperature* and *Humidity* may be the input attributes. *Fan Speed*, *Heater*, and *A/C Compressor* may be the output attributes. Unlike numerical variables, linguistic variables do not have numerical values. Each linguistic variable has a set of linguistic values, or terms. The linguistic variable *Temperature*, for example, may have terms COLD, COOL, WARM, HOT, and VERY HOT. Similarly, DRY, MILD, and DAMP may be terms of the linguistic variable *Humidity*. The output variables also have terms. OFF, LOW, MEDIUM, and HIGH may be terms of *Fan Speed*, for example.

The control task is described by a set of rules involving the linguistic variables and their terms. Let L1, L2, Ln be the linguistic variables, where Lk has the terms Tk1, Tk2,... The following format is used to describe the rules.

```
IF (L1 IS T1i) AND (L2 IS T2j) AND ... THEN (Lm IS Tmk)
```

Strictly speaking, the parentheses are not necessary, but are included to improve readability. Following the climate control example, a rule may be,

```
IF (Temperature IS COLD) AND (Humidity IS DAMP) THEN (HEATER IS HIGH),
```

or suppressing the parentheses,

```
IF Temperature IS COLD AND Humidity IS DAMP THEN HEATER IS HIGH.
```

Notice that each rule consists of a set of conditions, each in the format (Li IS Tij), combined with the keyword AND. Each rule also has an action in the same format (Lm IS Tmk), concatenated to the conditions with the keyword THEN.

Each term is modeled as a fuzzy set. A fuzzy set is a collection of elements. Each element has a membership grade between 0 and 1. The membership grade 0 implies that the element is not a member of the set, whereas, a 1 implies that the element is definitely a member of the set. Membership grade values between 0 and 1 describe a varying degree of membership. Terms, or fuzzy sets may be described graphically.

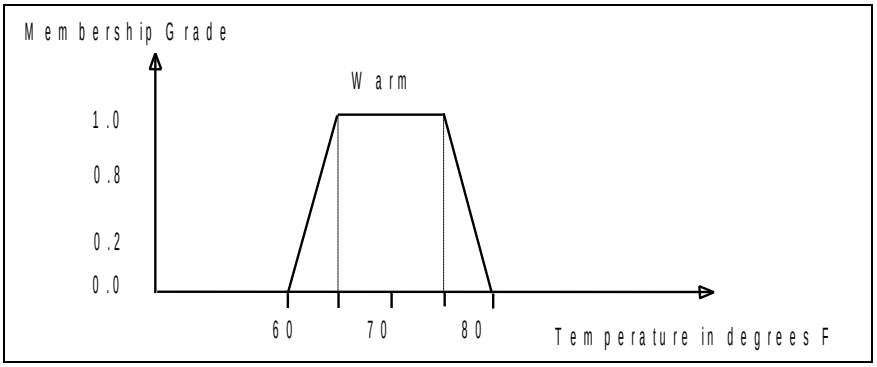

Figure 1. The Term "WARM" of the Linguistic Variable "TEMPERATURE"

The range from 65 to 75 degrees have membership grade 1, meaning the range 65 to 75 is definitely WARM. The range below 60 and above 80 have membership grade 0, meaning that these ranges are definitely NOT WARM. The range between 60 and 65 is the transitional area or "gray area" where it temperature gradually changes from NOT WARM to WARM. Similarly, the range 75 to 80 corresponds to the transition from WARM to NOT WARM. The membership function has a trapezoidal shape. Although in theory the membership grade function may have any shape, trapezoidal functions are by far the most used forms in FLC. Note that the trapezoidal membership function, and thus a term, may be described with 4 points. In this example, the term WARM may be described by {60, 65, 75, 80}. These 4 numbers are interpreted as follows: The inner 2 numbers [65, 75] is the interval definitely WARM. Any value below the first point [60] and any value above the fourth point [80] are definitely NOT WARM. That is, the four points may be viewed as the extension of the simple interval [65, 75] which now includes the "gray areas" or transitional regions.

Other forms of the trapezoidal functions, such as triangular functions, are possible. A single graph is used to show three terms of the linguistic variable *Temperature*.

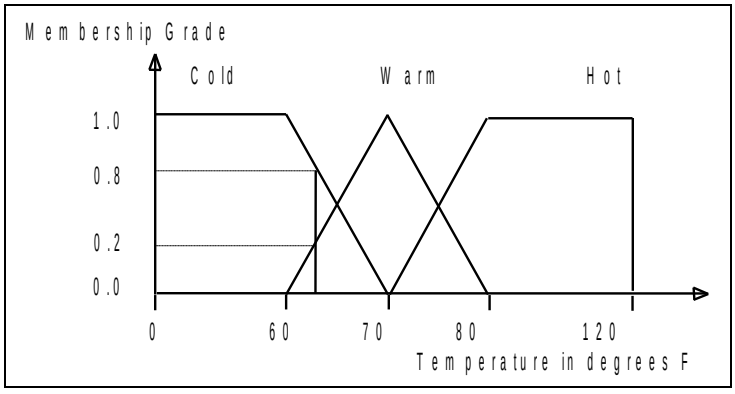

Figure 2. The Terms of the Linguistic Variable "TEMPERATURE"

In this example, WARM has a triangular membership grade function. Only the point 70 is definitely WARM. COLD has no transitional region at point 0, and similarly, HOT has no transitional region at point 120. The terms of Temperature are expressed as follows.

```
COLD = {0,0,60,70}
WARM = {60,70,70,80}
HOT  = {70,80,120,120}
```

Temperature may be measured by a thermistor connected to an Analog-to-Digital Converter (ADC). An 8-bit ADC will return values from 0 to FFh, not values in degrees. However, the user interface is more intuitive if ^{o}F scale is used. rFLASH keeps track of two scales or units. The first is referred to as the application coordinates, such as degrees. The system coordinates refer to the binary value stored internally. The application coordinate values are either bytes or words. Both signed and unsigned integers are supported. The sign integers are in two's complement form.

Similarly, the three terms of linguistic variable "*Humidity*" are given below, using the relative percent humidity scale.

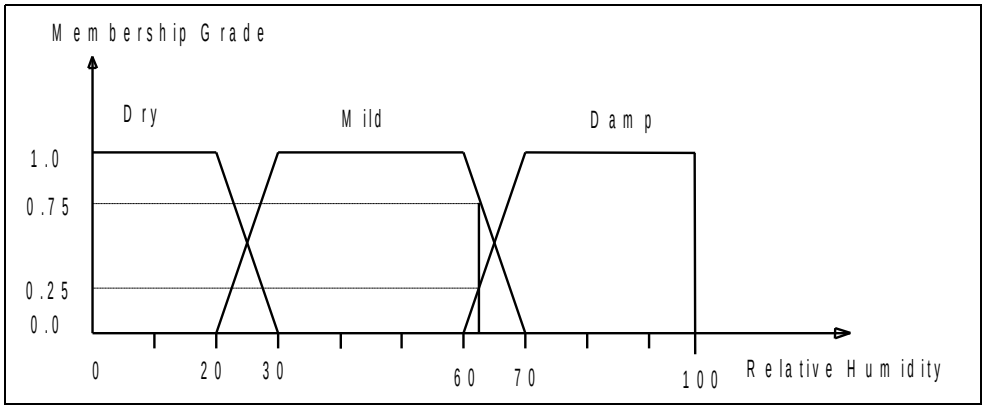

Figure 3. Terms of the Linguistic Variable "HUMIDITY"

In FLC, the outputs are usually described by single-valued terms, or singletons. The terms OFF, LOW, MEDIUM, and HIGH of *Fan Speed*, for example may be assigned values 0, 25, 50, and 100 percent.

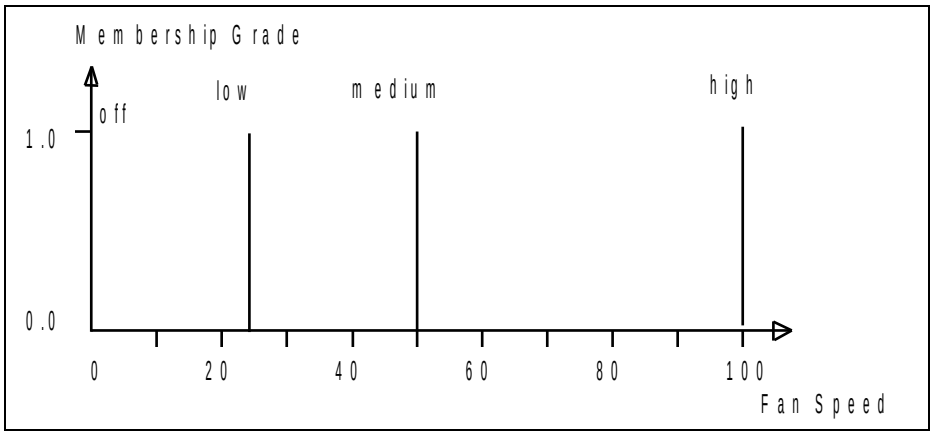

Figure 4. Terms of the Linguistic Variable "FAN SPEED"

Using the extended range format, these values are expressed as follows.

```
OFF    = {  0,  0,  0,  0}
```

```
LOW    = { 25, 25, 25, 25}
MEDIUM = { 50, 50, 50, 50}
HIGH   = {100,100,100,100}
```

This notation looks (and is) unnecessarily complicated, but it is mathematically correct. The term LOW, for example, states that the Fan Speed is definitely LOW within the interval [25, 25], the two inner points, and is definitely not LOW below 25 and above 25. Since all 4 points have the same value, rFLASH accepts a single value to be specified for output singletons.

```
OFF    = {  0}
LOW    = { 25}
MEDIUM = { 50}
HIGH   = {100}
```

Once the terms of the input and output variables are defined, the rules are stated. For the climate control task, consider the following rules.

```
if Temperature is  COOL and Humidity is DRY   then Fan_Speed  is OFF
if Temperature is  COOL and Humidity is DRY   then Heater     is LOW
if Temperature is  COOL and Humidity is DRY   then Compressor is OFF

if Temperature is  WARM and Humidity is DRY   then Fan_Speed  is LOW
if Temperature is  WARM and Humidity is DRY   then Heater     is OFF
if Temperature is  WARM and Humidity is DRY   then Compressor is OFF
```

Note that the rules are grouped in triplets. Each rule of the group has the same conditions, but different actions. In fact, there are three actions, one for each of the outputs *Fan Speed*, *Heater*, and *AC Compressor*. rFLASH allows an alternative format to reduce the size of the CTDF. In this alternative format, the actions are combined with commas as shown below.

```
if Temperature is  COOL and Humidity is DRY   then Fan_Speed  is OFF, Heater
      is LOW, Compressor is OFF

if Temperature is  WARM and Humidity is DRY   then Fan_Speed  is LOW, Heater
      is OFF, Compressor is OFF
```

Although the rules are grouped together, rFLASH expands these compound rules into individual rules, each with a single output action.

The output of the fuzzy logic controller depends on the rules, or more specifically, on the membership grades of the rules, depending on the current input values. Typically, there are two opposing requirements that need to be balanced in this step. First, the rule or rules with higher membership grades should be weighted more in constructing the output. Secondly, the effect of the rules with lower membership grades should nonetheless be taken into account to assure that the output switches smoothly from one value to another. This means that the weights given to the rules with higher membership grades need be allocated conservatively.

In FLC, first each condition is evaluated by finding the membership grade of each term of each linguistic variable. Then, the membership grade of each rule is evaluated by taking the minimum of the membership grades of the conditions. The singleton output of each rule takes the rule membership grade as a weight. If more than one rule have the same action (output singleton), the weight is the maximum of the membership grades. This is the inferencing stage, producing a set of singleton outputs, each with a membership grade. Computing the crisp outputs, or the defuzzification stage, is next. For each output, a numerical value must be computed. If the centroid method is used, the singleton outputs are averaged with their corresponding weights. The Centroid method computes the center of gravity of the fuzzy set. The set of singletons may be viewed as a fuzzy set. The centroid is then computed as

$$\frac{\sum_i S_i F_i}{\sum_i F_i}$$

where S_i is the singleton action value for rule R_i and F_i is the membership grade for rule R_i.

The Minimax method simply selects the output value which has the highest membership grade. Experience reveals that the centroid method, although computationally more intensive, gives smoother responses.

3 CONTROL TASK DESCRIPTION FILE FORMAT

The Control Task Description File (CTDF) is the input to rFLASH. It consists of five types of statements:

1. inputs
2. outputs
3. terms
4. rules
5. options

In addition, comments are allowed following either a semicolon (;) or a double slash (//). Statements are initiated by a pound sign (#) followed by one of the keywords input, output, term, rule, or define. rFLASH is case insensitive. The five types of statements are now described in further detail.

3.1 Input Statements

The format of input statements is

```
#input  type  identifier  media  address  [application coordinate range]
```

The type field refers to the variable type used to store the numerical value of the input variable. Four choices are available, BYTE, SBYTE, WORD, and SWORD. BYTE and WORD are 8-bit and 16-bit unsigned values, respectively. SBYTE (signed byte) and SWORD (signed word) refer to 8-bit and 16-bit integer values stored in two's complement notation.

The field identifier is the user-given name of the linguistic variable. The field media determines the source, either REGISTER or EXTERNAL. The field address is the internal or external data memory address of the source. When 16-bit integers of type WORD or SWORD are stored in memory, two consecutive memory bytes are used. The address is actually the address of the lower byte memory location. The high byte of the word is assumed to be in the lower address memory location. For example, for the input Humidity declared as

```
#input  word  Humidity     external  0C000H  [0..100]
```

the high byte is expected to be in external data memory at address 0C000h, and the low byte in 0C001h.

The optional field application coordinate range gives the lower and upper bounds. The format is [lowerbound..upperbound]. The application coordinate lower bound is mapped into the system coordinates depending on the variable type.

type	lower bound	upper bound
BYTE	0	FFh
SBYTE	80h	7Fh
WORD	0	FFFFh
SWORD	8000h	7FFFh

For example,

```
#input  word  Humidity     external  0C000H  [0..100]
```

declares Humidity to be an input linguistic variable which is to be read from external memory at address 0C000h whose value 0..0FFFFh is mapped into application coordinates range 0..100. More specifically, the application coordinate value 0 percent is internally represented by 0, and 100 percent by FFFFh.

Please note that when using signed integer types SBYTE and SWORD, if the origin of the application coordinates (value 0) is to coincide with the origin of the system coordinates, then the application coordinate range must be symmetrically set about 0. For example,

```
#input  sword  Temperature    register 0F0h  [-20..120]
```

associates the lower bound of the application coordinate range -20 with 8000h and the upper bound 120 with 7FFFh. The temperature 0 degrees corresponds to the system coordinate value 0A492h. Whereas, in the example given below,

```
#input  sword  Position       register 0C0H  [-100..100]
```

the upper bound 100 is associated with system coordinate value 7FFFh and the lower bound -100 with 8000h. Since the range is symmetrical about 0, the position 0 is represented by 0 in system coordinates.

3.2 Output Statements

The output statement format is similar to the input statement format, except that it starts with the keyword output. For example,

```
#output byte  Fan_Speed    external  0C002h   [0..100]
```

declares Fan_Speed to be an output linguistic variable which is to be written to external memory at address 0C002h whose value 0..0FFh is mapped into application coordinates range 0..100. More specifically, the application coordinate value 0 percent is internally represented by 0, and 100 percent by FFh.

3.3 Term Statements

Term statements have the format

```
#term identifier  (linguistic variable)   {point set}
```

where identifier is the user-given name of the term, linguistic variable is the name of the linguistic variable the term is associated with, and the point set is 4 points for terms of input linguistic variables, or a single point for output linguistic variables. Note that the name of the associated linguistic variable is written within regular parentheses, and the point set, within curly parentheses. For example,

```
#term  DRY   (Humidity)  {0,0,50,70}
```

declares the term DRY to be associated with linguistic variable Humidity with points 0,0,15, and 70 describing the trapezoidal membership function. Consider the term LOW for the output linguistic variable Fan_Speed.

```
#term LOW  (Fan_Speed)  {50}
```

Note that the point set consists of a single point.

3.4 Rule Statements

The format

#rule if condition1 and condition2 and ... and condition then action [,action[,action[...]]]

is used for rules, where condition i is in the form

InputLinguisticVariable is Term

and action is in the form

OutputLinguisticVariable is Term

For example, the rule

```
#rule if Temperature is COOL and Humidity is DAMP then Fan_Speed is LOW
```

states that Fan Speed should be LOW if Temperature is COOL and Humidity is DAMP. Several actions with the same set of conditions may be grouped together separated by commas. For example,

```
#rule if Temperature is COOL and Humidity is DAMP then
```

```
                        Fan_Speed is LOW, Heater is HIGH, Compressor is OFF
```

associates three actions to the conditions "if temperature is cool and humidity is damp." The rules with multiple actions are expanded by rFLASH and written as multiple rules, each rule with a single action.

3.5 Options Statements

The options statements are used to specify the defuzzification method, the internal data memory block address used during calculations, and the processor type. The format is given below.

```
#define [ centroid | minimax ]
#define data address
#define [ P8031 | P8032 ]
```

Centroid and minimax are the two defuzzification methods. The default is the centroid method. The defuzzification method may be changes many times in the CTDF, each defuzzification method being active until a new method is selected.

rFLASH uses a block of about 24 bytes of internal data memory. This block is specified by its starting address. The address of this block is declared following the keyword data. The default address is 0E8h. This option is ignored if the -e switch is used. When the -e switch is in effect, the generated code is assumed to run under READS, and the default data address 0E8h is used.

The processor type determines the size of internal data memory. Two options are available P8031 implying 128 bytes of memory, and P8032, 256 bytes of memory. The default is P8032 with 256 bytes of memory.

The CTDF for the climate control task example is given below.

```
// comment: Climate Control Example 5/14/93

; options
#define centroid
#define p8032

; inputs
#input  word  Humidity     external  0C000H  [0..100]

#input  word  Temperature  register  0F0h    [-20..+120]

#output byte  Fan_Speed    external  0C002h   [0..100]
#output byte  Heater       external  0C004h   [0..100]
#output byte  Compressor   external  0C006h   [0..100]

; all the terms consist of 4 data points {a, b, c, d} (trapezoidal shaped)

; terms for Humidity
#term  DRY   (Humidity)  {0,0,50,70}
#term  MILD  (Humidity)  {60,70,75,80}
#term  DAMP  (Humidity)  {70,90,100,100}

// terms for Temperature
#term  COLD    (Temperature)  {-20,-20, 40, 50}
#term  COOL    (Temperature)  { 40, 50, 50, 60}
#term  WARM    (Temperature)  { 50, 60, 70, 80}
#term  HOT     (Temperature)  { 70, 85, 95,100}
#term  VERYHOT (Temperature)  { 95,100,120,120}

// term for Fan_Speed (output singletons)
#term OFF     (Fan_Speed)   {0}
#term LOW     (Fan_Speed)  {50}
#term HIGH    (Fan_Speed) {100}
```

```
// term for Heater (output singletons)
#term OFF     (Heater)   {0}
#term LOW     (Heater)  {40}
#term MED     (Heater)  {70}
#term HIGH    (Heater) {100}

// term for air conditioning Compressor (output singletons)
#term OFF     (Compressor)   {0}
#term LOW     (Compressor)  {30}
#term MED     (Compressor)  {75}
#term HIGH    (Compressor) {100}

// definitions of rules
#rule if Temperature is  COLD                       then Fan_Speed  is HIGH
#rule if Temperature is  COOL                       then Heater     is HIGH
#rule if Temperature is  COOL                       then Compressor is OFF

#rule if Temperature is  COOL and Humidity is DRY   then Fan_Speed  is OFF
#rule if Temperature is  COOL and Humidity is DRY   then Heater     is LOW
#rule if Temperature is  COOL and Humidity is DRY   then Compressor is OFF

#rule if Temperature is  COOL and Humidity is MILD  then Fan_Speed  is LOW
#rule if Temperature is  COOL and Humidity is MILD  then Heater     is MED
#rule if Temperature is  COOL and Humidity is MILD  then Compressor is OFF

#rule if Temperature is  COOL and Humidity is DAMP  then Fan_Speed  is LOW
#rule if Temperature is  COOL and Humidity is DAMP  then Heater     is HIGH
#rule if Temperature is  COOL and Humidity is DAMP  then Compressor is OFF

#rule if Temperature is  WARM and Humidity is DRY   then Fan_Speed  is LOW
#rule if Temperature is  WARM and Humidity is DRY   then Heater     is OFF
#rule if Temperature is  WARM and Humidity is DRY   then Compressor is OFF

#rule if Temperature is  WARM and Humidity is MILD  then Fan_Speed  is LOW
#rule if Temperature is  WARM and Humidity is MILD  then Heater     is OFF
#rule if Temperature is  WARM and Humidity is MILD  then Compressor is MED

#rule if Temperature is  WARM and Humidity is DAMP  then Fan_Speed  is HIGH
#rule if Temperature is  WARM and Humidity is DAMP  then Heater     is OFF
#rule if Temperature is  WARM and Humidity is DAMP  then Compressor is MED

#rule if Temperature is  HOT                        then Fan_Speed  is LOW
#rule if Temperature is  HOT                        then Heater     is OFF
#rule if Temperature is  HOT                        then Compressor is HIGH

#rule if Temperature is  VERYHOT                    then Fan_Speed  is HIGH
#rule if Temperature is  VERYHOT                    then Heater     is OFF
#rule if Temperature is  VERYHOT                    then Compressor is HIGH
```

Figure 5. CTDF for the Climate Control Task Example.

4 RUNNING rFLASH

4.1 Setup

rFLASH is a Windows program and may be started from the Program Menu or an Icon placed on the desktop.

There are three compile options, for the V4 Toolchain, for Report files, and for RROS. The check box for the V4 Toolchain selects whether you wish to use our older absolute assemble and our version 3 Toolchain, or our relative assembler in our version 4 Toolchain. We recommend for all new applications you select our version 4 Toolchain. The check box for Report files causes rFLASH to generate a report file listing all inputs, terms, conditions, actions, rules, and outputs. The check box for RROS is convenient when the rFLASH-generated code is to be run on one of Rigel's evlauation and training boards with Reads51 (Rigel's Embedded Applications Development System). Selecting RROS instructs Reads51 to use the support subroutines placed in the board EPROM. This reduces the size of the rFLASH-generated code, which simplifies downloading and debugging the applications programs under Reads51

There are two Simulator settings which set up the rFLASH simulator. The Simulator Tabulate Check Box turns the Simulator on and the Step Size % asks for how much detail you wish to view during simulation. You may select any where from 10% to 100%.

The report file and the simulator are intended to assist in constructing the terms and rules.

If the Simulator is not selected, provided that there are no errors, rFLASH generates an assembly language file with the name filename and extension .asm. In case there are error, rFLASH generates a list file with the name filename and extension .lst. The list file shows the errors encountered.

4.2 Errors

The errors are classified into syntax errors and semantic errors. Syntax errors are where rFLASH could not decode the statement due to typographical errors or incorrect statement formats. For example,

```
#term  DRY   (Humidity)  {0,0,50,70z}
```

would cause a syntax error, since 70z is not a number.

Semantic errors are when rFLASH could not make sense of the statements. For example,

```
#term  DRY   (Humidity)  {0,0,70,50}
```

causes a semantic error since, although the syntax is correct, the 4 points must be in nondecreasing order.

4.3 Microcontroller Resources Used

Besides a block of internal data memory, rFLASH uses only one register bank. You may specify the register bank to be used by the "bankN" directive, where N is 0 to 3. If the register bank is not specified, the currently selected register bank is used. For example,

```
#define bank0
```

uses the register bank 3. If a register bank is specified, the currently used register bank is saved by pushing PSW on stack. This register bank is re-selected after the iteration by popping PSW.

rFLASH uses the stack to store data and for calls, depending on the data types and on the defuzzification method. In the worst case, 32 bytes of stack is used.

By default, the word variables (16-bit inputs and outputs) are stored in a big-endian fashion. That is, the high byte of the word is stored in memory with the lower address. Note that most C compilers, including the Reads51 C compiler, use the little-endian ordering. The following two directives are provided:

```
#define little_endian
#define big_endian
```

The default is big_endian to maintain compatibility with the older version of the software.

4.4 Using the Generated Code

The subroutine generated filename.asm contains two user callable subroutines: FLASHPoll and Output. FLASHPoll computes all output values and stores them into the internal or external memory addresses specified by the input and output statements. Sometimes it is desired to compute only one of the outputs. The subroutine Output should be used in this case. The output whose value is to be computed must be specified as follows:

```
mov   dptr, #OLV
lcall output
```

where OLV is the name of the user-given output linguistic variable. For example,

```
mov   dptr, #Fan_Speed
lcall output
```

will compute and store the output value for the output Fan_Speed.

The code generated by rFLASH must be supplemented by user code to obtain the values of the input and place them into the internal or external data memory locations, and then to read the results computed from the internal or external data memory locations. The user program is then expected to use the output values depending on the specific application.

The code generated by rFLASH and user-written code may be merged and assembled. We recommend you use Reads51 and assemble the code generated by rFLASH, allowing subroutines Output and rFLASHPoll to be accessed from other modules. User-written code may then be assembled and linked.

5 THE REPORT FILE

The report file is generated when the check box for Report is selected. The report file lists all the inputs, terms, conditions, rules, actions, and outputs. Typically, the report file is not necessary to generate FLC applications with rFLASH. It is supplied to assist in debugging the application or to be used as a summary document. The report file has the following format.

Terms
Input Linguistic Variables
Conditions
Rules
Actions
Output Linguistic Variables

5.1 Terms

All terms of the input and output linguistic variables are enumerated from 0 to n-1, where n is the total number of terms. The number corresponding to a given term is referred to as its code number or term code. For each term, the name, the associated linguistic variable, the number of points in the membership grade function, and the points in both application coordinates and in system coordinates are given. Two samples are shown below.

```
term 3
--- Term ---
               name : COLD
 source line number : 24
linguistic variable : Temperature
   number of points : 4
points (app. coord.):     -20     -20      40      50
points (sys. coord.):   00000h  00000h  06DB6h  07FFFh

term 16
--- Term ---
               name : ON_LOW
 source line number : 43
linguistic variable : Compressor
   number of points : 1
points (app. coord.):      30
points (sys. coord.):   0004Ch
```

5.2 Input Linguistic Variables

All input linguistic variables are enumerated from 0 to n-1, where n is the total number of input linguistic variables. For each input linguistic variable, the name, the source line number where the variable is defined, the type, the media, the input address and the application coordinate range are given. The range is first shown in hexadecimal, then in decimal. A sample is shown below.

```
input variable 1
--- Input Linguistic Variable ---
               name : Temperature
 source line number : 10
               type : word
              media : register
      input address : 000F0h   (240)
  app. coord. range : [0FFECh .. 00078h]  ( [-20 .. 120] )
```

5.3 Conditions

All conditions are enumerated from 0 to n-1, where n is the total number of conditions. For each condition, the linguistic variable name and the term name are given. In addition, each condition is given an identification number, referred to as its code or code number. Similarly, the term code corresponding to the term is given. The input address of the linguistic variable is given both in hexadecimal and in decimal. A sample is shown below.

```
condition 4
--- Condition ---
linguistic variable name : Temperature
               term name : WARM
                    type : 41h
                    code : 4
               term code : 5
           input address : 000F0h   (240)
```

5.4 Rules

All rules are enumerated from 0 to n-1, where n is the total number of rules. A code number is given to each rule. For each rule, the number of conditions of the rule, and the number of actions are given. rFLASH expands the compound rules with multiple actions to conform with FLC. Each rule thus has a single action after expansion. The singleton value for the action is given in both application coordinates and in system coordinates, both in hexadecimal and in decimal. Next the codes of the conditions and of the actions are listed. A sample is shown below.

```
rule 6
--- Rule ---
               code : 6
   no of conditions : 2
   no of    actions : 1
   singleton output
        app. coord. : 00032h  (50)
        sys. coord. : 0007Fh  (127)
 source line number : 57
    condition codes :    0  3
       action codes :    3
```

5.5 Actions

All actions are enumerated from 0 to n-1, where n is the total number of actions. A code number is given to each action. For each action, the output linguistic variable name, the term name, and its code is given. Moreover, the number of rules that have the action followed by the codes of the rules with the action is presented. A sample is shown below.

```
action 2
--- Action ---
linguistic variable name : Compressor
               term name : OFF
                    code : 2
             no of rules : 4
              rule codes :   2  5  8  11
```

5.6 Output Linguistic Variables

All output linguistic variables are enumerated from 0 to n-1, where n is the total number of output linguistic variables. For each output linguistic variable, the name, the source line number where the variable is defined, the type, the media, the output address and the application coordinate range are given. The range and address are first shown in hexadecimal, then in decimal. Finally, the codes of the actions using the output linguistic variable are listed. A sample is shown below.

```
output variable 1
--- Output Linguistic Variable ---
               name : Heater
 source line number : 13
               type : byte
              media : external
  app. coord. range : [00000h .. 00064h]  ( [0 .. 100] )
     output address : 0C004h   (49156)
       action codes :   1  4  5  6
```

6 THE rFLASH SIMULATOR

The simulator is invoked when the Tabulator check box is selected.. Note that when the simulator is invoked, no assembly file is generated. The simulator is intended to be used while constructing and fine-tuning the terms and rules of the FLC application. Once the terms and rules are set, rFLASH is run without the simulator to generate the assembly file containing the subroutines.

7 BIBLIOGRAPHY

Books on General Fuzzy Set Theory

1. *Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh*, Edited by R. R. Yager, Wiley-Interscience, 1987.
2. *Fuzzy Sets and Systems: Theory and Practice*, by D. Dobois and H. Parade, Academic Press, 1980.

Books on Fuzzy Logic Control

3. *Neural Networks and Fuzzy Systems* by B. Kasko, Prentice Hall, 1991.

Articles on Fuzzy Logic Control

8. Thinking Clearly with Fuzzy Logic, by R. Leigh and M. Wetton, *Process Engineering*, November 1983.
7. Software and 'Fuzzy' Logic Let Any Good Programmer Design an Expert System, by A. Okuma, *Electronic Design*, April 1985.
5. Fuzzy Logic Simplifies Complex Control Problems, by T. Williams, *Computer Design*, March 1991.
4. Creating Fuzzy Micros, by J. M. Sibigtroth, *Embedded System Programming*, December 1991.
6. Fuzzy Logic is Anything but Fuzzy, by T. Williams, *Computer Design*, April 1992.
9. Twenty Years of Micros- Now What?, by T. Cantrell, *The Computer Applications Journal*, volume 26, 1992.
10. "Fuzzy Logic," *Scientific American*, volume 269, number 1, July 1993.