Rapid Prototyping Methodology and Environments for Fuzzy Applications

Chantana Chantrapornchai

Faculty of Science, Silpakorn University, Nakorn Pathom 73000, Thailand
ctana@su.ac.th

Abstract. In this paper, we present an integrated design environment and methodology for fuzzy applications. The framework integrates the concept of rapid system design where the given system specification is broken into two components: *conceptual* and *parameter* specifications. The *conceptual specification* contains the system core, which is unchanged throughout the simulation phase and can be implemented in hardware at an early stage to create partial prototype. The *parameter specification* includes variables that are adjustable for system performance tuning. We develop a design environment to facilitate fuzzy system designers to verify the correctness of the system behavior by varying their parameters. A design example using our framework is demonstrated.

1 Introduction

In the design of application specific systems, the iterative process of detailing specifications, designing a possible solution, and testing and verifying the solution is repeatedly performed until the solution meets user requirements. However, such a repeated cycle prolongs the system development time. The system life cycle is likely to be short due to rapid pace of improvements in current technology. It is likely that the system will be out of date by the time it is ready to be shipped unless the development time is improved. The need for early prototyping, therefore, becomes critical to implement a system specification and provide customers with feedback during the design process.

Embedding all system design components into hardware is not a good approach due to lack of flexibility during reconfiguration. In fuzzy systems, by partitioning the system components properly, an early prototype can be established [1]. In this paper, our methodology and design environment integrate hardware and software capability and provides flexibility in exploring several design solutions. Given a set of user requirements, a system specification which describes what system functions are needed to satisfy the requirements is developed. Such a specification can be abstractly divided into two parts: a *conceptual specification* and a *parameter specification.* The conceptual specification contains a specification core that is rarely changed during various system adjustment while the parameter specification includes specific details which can be easily modified during the design phase. We are developing an integrated design environment which facilitate designers to design fuzzy systems based on the above concept. The framework enables designers to test many possibilities of selecting parameters. The built-in simulation feature allows designers to verify the system behavior for the tested parameters.

Considerable work has been done on developing hardware implementations for general purposed fuzzy control systems. In [2], special hardware chips have been invented to speedup the fuzzy inference engine. Other work has concentrated on design and implementation of fuzzy

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 940–949, 2003.

architectures and processors [3, 10]. Software fuzzy systems are flexible but fail to provide high-speed result [7]. Although general-purposed fuzzy hardware can yield high-speed output, they may be able to neither meet stringent real-time requirement of specific applications nor specific needs of the applications. Kan and Shragowitz presented presented a generic development tool for fuzzy systems [4]. Their tools provide flexibility in defining membership functions, rules, and fuzzy logic operations. Nevertheless, they do not focus on constructing hardware modules from the derived system. In [5], a computer-aided design tool for implementation of fuzzy controllers on FPGAs was proposed. However, they do not provide flexibility during a prototyping phase,.e.g., membership functions are implemented as a boolean circuit. Also, none of these works provide a flexible framework and tools which support designing systems with many input variables and various shapes of membership functions as well as testing environments for both crisp and fuzzy inputs. Unlike the others, our research provides rapid prototyping framework for application specific fuzzy systems as well as flexible implementation.

Since one of the goals of using fuzzy logic is to reduce computational complexity of designing systems, e.g., fuzzy control systems, fuzzy logic rule bases consist of human knowledge expressed in rule format. In our approach, the conceptual specification consists of the rule base portion since it defines system functionality and is seldom changed or required only a few modification. In some classical fuzzy systems which have practical applications such as temperature controller, and inverted pendulum, the rules can commonly be found [9]. For detailed fuzzy system tuning, varying parameters are considered such as the range of system operation, the shape of membership functions, type of fuzzy logic operations, defuzzification methods, etc. We collectively include these portions in the parameter specification. Such a specification can be implemented in software for flexible tuning.

The details of our method will be discussed in the remainder of this paper. The model and synthesis methodology are discussed in Section 3. Section 4 presents an example of prototyping a fuzzy control system. Finally, Section 5 draws conclusion from this work.

2 Backgrounds

In this section, we provide some background knowledge related to fuzzy system characteristics. Fuzzy set originated by Zadeh [11] differs from the traditional crisp set. For crisp sets, the set boundary is crisp. For example, let A be the set of real numbers ranges in $[0, 10]$. We can determine whether or not a given number is a member of the set or not. We may define a membership function called $\mu_A(x)$ which maps from a real number x to a value in $[0, 1]$. Given a set $Y \subset X$, where $Y = [5, 8]$. The membership value of $y \in Y$ for $\mu_X(y)$ is 1 for all y.

For fuzzy sets, the boundary of the set is uncertain. Thus the fuzzy set is used to represent another kind of uncertainty. For example, let the universe of the age of people be $[1..20]$. Let B be a set of "young" people. Since young is a linguistic variable, the meaning of "young" varies depending on the interpretation of experts. Once people is 20 years old, will he definitely not a young people anymore? This cannot be determined precisely. Therefore, a set of young people may represent a set with fuzzy boundary. For example, for a people whose age is 25 may be considered as a young people with the degree of 25.

For simplicity, we assume that inputs are fuzzy sets and a given set of rules is in a canonical form and each rule yields only a single consequence.

Given any two inputs i and j, applying the Max-Min inference, we compute the cut of the output membership degree as shown in Figure 2(a) and Figure 2(b). After a cut is computed for each output membership function, all of them are unioned together as shown in Figure 2(c). Then, a defuzzification method is employed to change the output to crisp a value.

A general form of a rule comprising two or more linguistic variables conjunctive ($\wedge$) to each other is in the following form:

$$R_1\text{: IF } x_1 \text{ is } X_{11} \wedge \ldots \wedge x_m \text{ is } X_{1m}, \text{ THEN } y \text{ is } Y_1$$
$$R_2\text{: IF } x_1 \text{ is } X_{21} \wedge \ldots \wedge x_m \text{ is } X_{2m}, \text{ THEN } y \text{ is } Y_2$$
$$\vdots$$
$$R_1\text{: IF } x_1 \text{ is } X_{n1} \wedge \ldots \wedge x_m \text{ is } X_{nm}, \text{ THEN } y \text{ is } Y_n$$

The above system will require m non-interactive fuzzy set inputs, $\{X_1', X_2', \ldots X_m'\}$ and produce a single fuzzy set output Y'. X_{ij} is a fuzzy subset of corresponding universe of discourse U_j and Y_j is a fuzzy subset of the universe of discourse V. An element x_j denotes a fuzzy element in the related set X_{ij} and $\mu_{X_{ij}}(x_j)$ represents its corresponding membership value.

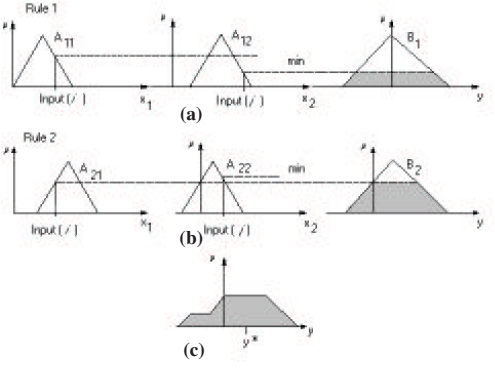


Fig. 1. Graphical Max-Min inference for two rules.

3 Design Methodology

In our methodology, we divide the specification into two parts: Conceptual and Parameter specifications.

3.1 Conceptual Specification

For the conceptual specification, it is the core component which represent the main behavior of the system. In fuzzy systems, we regard a rule base as a conceptual specification. The rule-base specification is modeled as a *Conceptual State Graph* which presents state-oriented behaviors of the rules.

Definition 1. *A Conceptual State Graph (CSG)* $G = (\mathbf{S}, \mathbf{I}, \mathbf{O}, \mathbf{E}, \omega)$ *is a connected directed edge-weighted graph where* $\mathbf{S}$ *is a set of nodes* $s_i \in S, 1 \leq i \leq p$*, representing states of the graph,* $\mathbf{I}$ *is the set of inputs,* $\mathbf{O}$ *is the set of outputs including* ϕ*,* $\mathbf{E} \subseteq S \times S$ *is a set of edges, denoted by* $s_i \rightarrow s_j$ *or* e_{s_i,s_j}*,* $s_i \in \mathbf{S}$*,* $s_j \in \mathbf{S}$*, and* ω *is a function from* $\mathbf{E}$ *to* $\mathbf{I} \times \mathbf{O}$*, representing the set of edge weights, denoted by* x/y*, where* $x \in \mathbf{I}$ *and* $y \in \mathbf{O}$*.*

For the edge weight x/y of an edge $e_{u,v}$ or $u \rightarrow v$, if $y = \phi$, then no output is given for the edge from u to v with the input x. As an example, $\mathbf{I}$ contains all combinations of values for x and y according to the rules, that is $\mathbf{I} = \{(x, L), (x, M), (x, H), (y, H)\}$. Similarly, $\mathbf{O} = \{(z, L), (z, H), \phi\}$, $\mathbf{S} = \{s_0, s_1, s_2, s_3\}$ and $\mathbf{E} = \{e_{s_0,s_1}, e_{s_0,s_2}, e_{s_0,s_3}, e_{s_1,s_3}\}$ Edges to states s_2 and s_3 have output labels (y, L) and (y, H) respectively. The output label for the intermediate edge (s_0, s_1) is ϕ. The edge weights are: $\omega(s_0, s_1) = (x, M)/\phi, \omega(s_0, s_2) = (x, L)/(z, L), \omega(s_0, s_3) = (x, H)/(z, H), \omega(s_1, s_3) = (y, H)/(z, H)$

Using this model, one can easily implement a system with multiple input variables. This can be advantageous since many current fuzzy processors allow only limited number of input variables. The CSG model can be transformed to an incompletely specified FSM and therefore, transformed to a completely specified FSM to which traditional FSM synthesis can be applied.The CSG is similar to the incompletely specified FSM, based on an assumption that no unspecified next state is encountered. All unspecified outputs, ϕ, convey "don't care" outputs and the set of final states are the set of states where outputs are given.

Based on the model, assume that the rule is in the canonical form, the number of bits used to represent output variables are $\lceil \log_2(Q + 1) \rceil$ where Q is the number of distinct output linguistic variables. The number of bits used to represent input variables is $\lceil \log_2(\sum_{j=1}^{N} M_j + 1) \rceil$ where M_j is the number of distinct input linguistic variables of input j and N is the number of inputs. Further the number of bits used to represent states equals to $\lceil \log_2(1 + Q + \sum_{j=1}^{N-1} M_j^j) \rceil$.

3.2 Parameter Specification

A parameter specification includes membership functions for input and output linguistic variables as well as defuzzification methods. For a membership function, a designer may vary the shape of membership functions and boundary. Figure 2 shows possible membership function shapes for a linguistic variable. A designer may specify different defuzzification methods such as centroid, weighted-average, Mean-Max etc. After a membership function shape and boundary are selected for each linguistic variable, a membership value is converted to discrete values ranged [0,255]. It is then mapped to a memory mapped table. The number of entries of this lookup table for each linguistic variable is P where P is the number of possible input values of each membership function that is greater than 0. Hence, totally we require the space $O(\sum_{j=1}^{N} \sum_{i=1}^{M_j} k_i + \sum_{i=1}^{Q} q_i)$ where k_i is the number of distinct values for each linguistic variable for input j and q_i is the number of distinct values for each output linguistic variable, Q is the number of output linguistic variables.

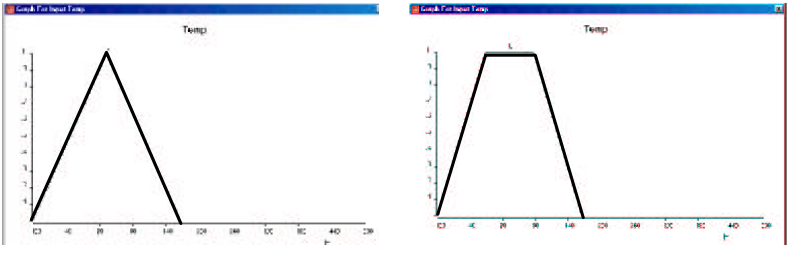


Fig. 2. Different shapes of membership functions (a) Triangular (b) Trapezoid

3.3 Component Integration

According to the fuzzy inference process, a special calculation is required to determine the output strength of each rule given the associated input strengths. We assume min operation is used for computing fuzzy rule strength. Let $\mu_{X_{ij}}(x_j)$ denote a membership function corresponding to the linguistic variable X_{ij} and the current input instance x_j for rule i. Let $\mu_{Y_i}(y)$ be a membership function corresponding to the linguistic variable Y and output instance y for rule i. Suppose a rule requires m inputs. Given input instance $x_1, \ldots, x_n$, for each rule i, we determine a modified output function $\mu'_{Y_i}(y)$ that is used in the defuzzification process, where $\mu'_{Y_i}(y) = \mu_{Y_i}(y)$ if $\mu_{Y_i}(y) < r_i$ and $\mu'_{Y_i}(y) = r_i$, otherwise. where the rule strength r_i, calculated by $r_i = \min_{j=1}^{n} \mu_{X_{ij}}(x_j)$ which is the limit of the output strength Y_i. If more than one rules give the same linguistic variable output Y_i and therefore generates a new function $\mu'_{Y_i}(y)$, each $\mu'_{Y_i}(y)$ is combined by using the max operation. $\mu''_{Y_i}(y) = \max_{\forall Y_k = Y_i}(\mu'_{Y_k}(y))$ Based on these and the given CSG, the following algorithm presents the overall inference process.

Algorithm 1 (InferenceCSG)
Input: *CrispInput* $\{I_j\}$
Output: *CrispOutput*

```
1  begin
2    set 1 to all rows of Cut
3    foreach row i of A  do
4      clear Reg to 0
5      foreach column j of A do
6        read CrispInput I_j and compute Degree using InputLinguistic
7        send A[i, j] to Rule CSG and get OutIndex
8        Reg = Compute MIN(Reg,Degree) od
9      Cut[OutIndex] = MAX(Cut[OutIndex],Reg)  od
10   CrispOutput = DEFUZZIFY(Cut,OutputLinguistic);
11 end
```

Algorithm 1 is based on the max-min inference. In particular, it presents the function of the control unit in Figure 3. When the inputs to the system are fuzzy rather crisp values, we add the component to compute the minimum cut between each fuzzy input and linguistic variable before computing the min operator. Memory A temporarily stores all combinations of input sequence to be fed to the Rule CSG. The size of memory A is $b\Pi_{i=1}^{N} M_i$ bits where b is the number of bits used to represent all states and N is the number of input variables and M_i is the number of input linguistic variables for each input i.

For each row i of A, each input variable j is fed to CSG. Reg is used to keep minimum values for each rule (Line 8). This is then used to be input to MAX component to compute the maximum value of each output linguistic variable (Line 9). Note that Line 6 and Line 7 can be done at the same time, i.e, computing input degree and computing degree of output linguistic variable. At Line 10, the defuzzification method is then invoked using the stored Cut and output membership function for each output linguistic variable. The following summarizes the size needed for memory used to store each part.

InputLinguistic	OutputLinguistic	InputGenerator
$c\sum_{j=1}^{N}\sum_{i=1}^{M_j} k_i$	$c\sum_{i=1}^{Q} k_i$	$b\Pi_{i=1}^{N} M_i$

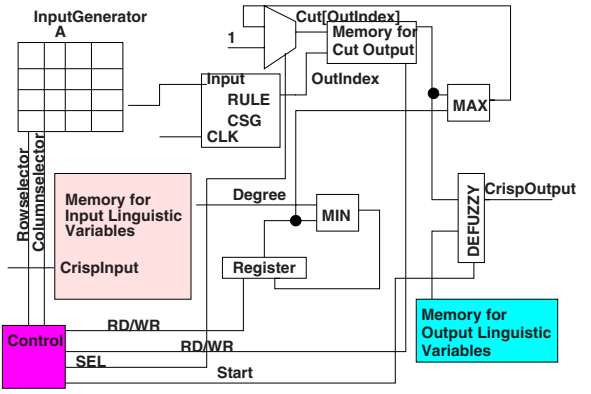


Fig. 3. Abstract connection of conceptual and parameter specification integration.

4 Design Example

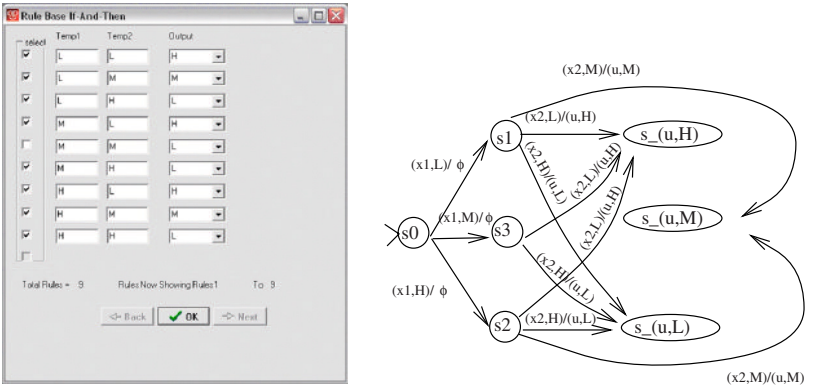
In this section, we show our design framework mapped to an example of a temperature control system for the oven found in [8]. We first create an initial design for a common rule base for the controller easily. In a meantime, we can test various parameters for the system based on our design environment.

4.1 Conceptual Specification

In the electric oven, the interior temperature is controlled by varying the heat input u to the jacket. Here two input variables are defined as the excess of temperature over the exterior, x_1, and x_2. Thus, the control system has two temperature input variables x_1 and x_2 and outputs a new temperature u so that the oven temperature is adjusted properly. Figure 4(a) uses our IDE to represent the following set of rules. In the figure, Temp1 is variable x_1 and Temp2 is variable x_2 while Output is u.

1. IF $x_1 = low\ (x_1, L)$ **and** $x_2 = low\ (x_2, L)$ THEN $u = high\ (u, H)$
2. IF $x_1 = low\ (x_1, L)$ **and** $x_2 = medium\ (x_2, M)$ THEN $u = medium\ (u, M)$
3. IF $x_1 = low\ (x_1, L)$ **and** $x_2 = high\ (x_2, H)$ THEN $u = low\ (u, L)$
4. IF $x_1 = medium\ (x_1, M)$ **and** $x_2 = low\ (x_2, L)$ THEN $u = high\ (u, H)$
5. IF $x_1 = medium\ (x_1, M)$ **and** $x_2 = high\ (x_2, H)$ THEN $u = low\ (u, L)$
6. IF $x_1 = high\ (x_1, H)$ **and** $x_2 = low\ (x_2, L)$ THEN $u = high\ (u, H)$
7. IF $x_1 = high\ (x_1, H)$ **and** $x_2 = medium$ d (x_2, M) THEN $u = medium\ (u, M)$
8. IF $x_1 = high\ (x_1, H)$ **and** $x_2 = high\ (x_2, H)$ THEN $u = low\ (u, L)$

The tuple next to each clause is a shorthand notation of the clause's linguistic variable. Figure 4(b) presents the CSG for this set of rules where the system inputs are temperatures x_1 and x_2 and the temperature output is u. An edge label is a tuple (x, z) where x is either variable x_1, x_2, or u and z is a linguistic variables with respect to $domain(x_1)$, $domain(x_2)$, or $domain(u)$ respectively.



(a) (b)

Fig. 4. (a) Rule specification based on our IDE. (b) CSG.

```
package input_output_type is
type data_3 is array (2 downto 0) of bit;
type data_2 is array (1 downto 0) of bit;
constant zero_2 : data_2 := "00";
constant zero_3 : data_3 := "000";
end input_output_type;
use work.input_output_type.all;
entity csg is
    port (input: in data_3;clk : in bit; s : out data_2; output : out data_2);
end csg;
architecture behavior of csg is
    SIGNAL state : data_2 := zero_2;
    SIGNAL statenext : data_2 := zero_2;
begin
rule: PROCESS (input,state, clk)
BEGIN
WAIT UNTIL (clk'event and clk = '1');
IF state = "00" and input = "000" THEN
    statenext <= "00";
    output <= "11";
elsif state = "00" and input = "001" THEN
    statenext <= "10";
    output <= "11";
elsif state = "00" and input = "010" THEN
    statenext <= "01";
    output <= "11";
elsif state = "01" and input = "011" THEN
    statenext <= "00";
    output <= "00";
.
.
.
else
    statenext <= "00";
    output <= "11";
end if;
state <= statenext;
END process rule;
end behavior;
```

Fig. 5. CSG VHDL implementation.

Figure 4(b) shows the CSG where S is $\{s_0, s_1, s_2, s_3, s_{(u,M)}, s_{(u,L)}, s_{(u,H)}\}$. The transitions to states $s_{(u,M)}$, $s_{(u,L)}$, and $s_{(u,H)}$ output temperature values *medium*, *low*, and *high* respectively. Figure 5 shows the VHDL representation of this CSG. In this implementation, linguistic variables $(x_1, L), (x_1, M), (x_1, H), (x_2, L), (x_2, M), (x_2, H)$ are represented by 000, 001, 010, 011, 100 and 101 while output variables $(u, H), (u, M), (u, L), \phi$ are encoded as 00, 01, 10 and 11 respectively. The initial state s_0 is encoded as 0. States s_1, s_2, and s_3 are encoded as 1, 2, and 3 respectively. After optimization, states $s_{(u,H)}$, $s_{(u,M)}$ and $s_{(u,L)}$ can be merged with the

initial state to save one control step and be ready to accept the next input. Thus, we only use 2 bits to encode existing states in this design. Any sequence of inputs that does not fire any rule will lead to the initial state.

4.2 Parameter Specification

The parameter specification includes membership functions and defuzzification approaches. Since all input and output variables are temperatures, the same membership functions for all variables are used. The temperature domain is restricted to between -100 and 500 degrees centigrade using our IDE. Then we may adjust the boundary of each input and output functions as shown in Figure 6.

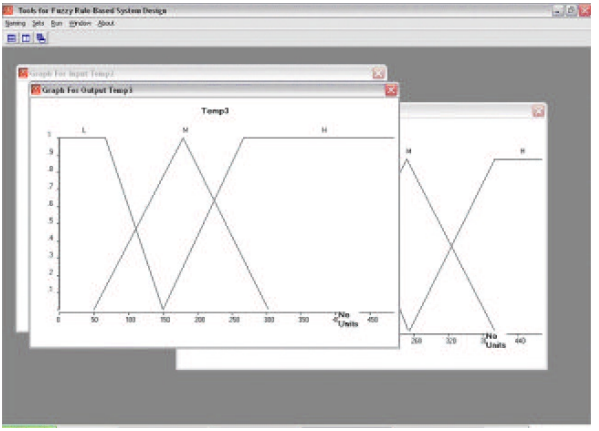


Fig. 6. Adjusting the boundary of membership functions.

For defuzzification method, we may have a choice of using centroid method, weighted average method, or Mean-Max method [8]. We may try to select each choice using our tool. In this example, we use the following equation. $x_1(t+1) = -2x_1(t)+x_2(t)+u(T), \qquad x_2(t+1) = x_1(t) - x_2(t)$ to compute the feedback function. where initial $x_1 = 90$ and $x_2 = 100$. Assume that membership function in Figure 6 is used and the centroid is used as a defuzzification method.

Using our IDE, the case when inputs are fuzzy values can also be simulated. Designers may test the system again fuzzy input cases when the shapes are varied as shown in Figure 8. Based on our framework, one can easily adjust the parameter parts to investigate design choices and re-simulate the system. Once the parameter is finalized, the prototype can be set-up rapidly.

Figure 9 shows the connection of the components and the size of each memory. For easy implementation, the membership values are scaled from real values ranging in $[0, 1]$ to integer values between $[0, 256]$. Input generator computes combinations of input linguistic variables which will be given to CSG rules (whose implementation is shown in Figure 5). The component Feedback this picture represents the feedback calculation where control action u together with inputs are used to calculate the new inputs for the next cycle. Here we use 16-bit address and data buses. The number of entries for input linguistic variables is 1806 and for output linguistic variable is 813. Input sequence generator A has 9 combinations, each of which is of three bits since we use 3 bits to represent all input linguistic variables. The size of OutIndex is 2 bits since only two bits is used to represents 4 possible output linguistic variables.

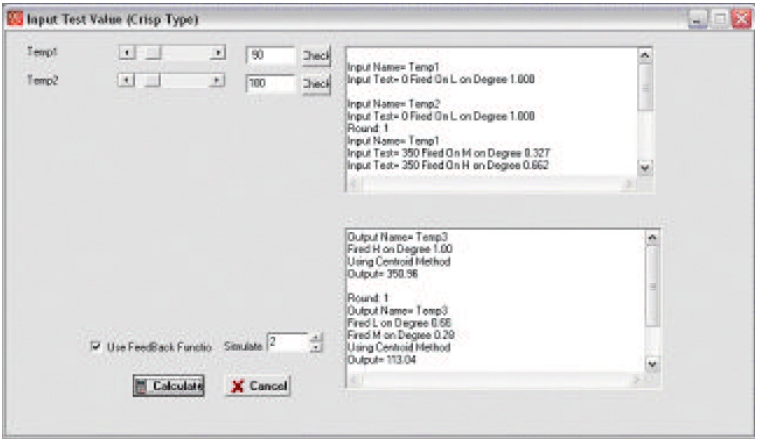


Fig. 7. Simulation for $x_1 = 90$ and $x_2 = 100$

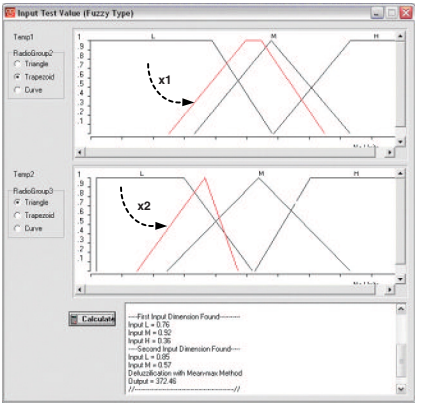


Fig. 8. Simulation for fuzzy inputs x_1 and x_2

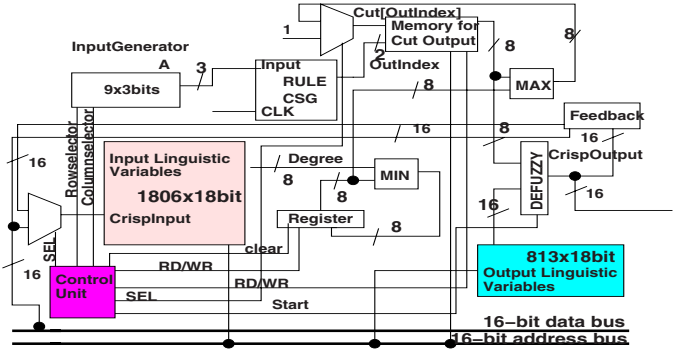


Fig. 9. Integration of all components.

5 Conclusion

We present a design methodology and environments for designing fuzzy controllers.Our methodology partitions system components into conceptual and parameterized specifications. The conceptual specification defines the core of system specification which is rarely changed and may be implemented as a hardware prototype to create a fast prototype. Parameterized components are specified in software in order to be easily edited for system modification. This method actually integrates hardware and software capability, yielding flexibility in re-specification and shorten prototyping time. We provide the IDE which gives flexibility for designers to explore fuzzy design parameters rapidly. By varying membership functions including shape and universe boundary as well as defuzzification methods, a designer performs various simulations to select the best possible parameters for the system. Altogether our framework enables rapid prototyping for fuzzy systems.

References

1. C. Chantrapornchai, S. Tongsima, and E. H. Sha. Rapid prototyping techniques for fuzzy controllers. In *Lecture Notes in Computer Science–ASIAN'99*, pages 37–49, 1999.
2. J. W. Fattaruso, S. S. Mahant-Shetti, and J. B. Barton. A fuzzy logic inference processor. In *Proc. of 3rd Intl. Conf. on Industrial Fuzzy Control and Intelligent Systems*, pages 210–214,1993.
3. A. Jaramillo-Botero and Y. Miyaka. A high-speed parallel architecture for fuzzy inference and fuzzy control of multiple processes. In *Proc. of theInternational Conference on Fuzzy Systems*, pages 1765–1770, 1994.
4. M. S. Khan and E. E. Swartzland Jr. Rapid prototyping fault-tolerant heterogeneous digital signal processing systems. In *Proc. of 6th Intl. Workshop on Rapid System Prototyping*, pages 187–193, 1995.
5. M. A. Manzoul. CAD tool for implementation of fuzzy controllers on FPGAs. *Cybernetics and Systems*, pages 599–609, 1994.
6. G. D. Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, Inc, 1994.
7. J. Moore and M. A. Manzoul. An interactive fuzzy CAD tool. *IEEE Micro* , pages 68–74, April 1996.
8. T. J. Ross. *Fuzzy Logic with Engineering Applications*. McGrawHill, 1st edition, 1995.
9. M. H. Smith. Tuning membership functions, tuning *and* and *or* operations, tuning defuzzication: which is the best? In *Proc. of the North American Fuzzy Information Processing Society Biannual Conference*, pages 347–351, 1994.
10. H. Watanabe. Risc approach to design fuzzy processor architecture. In *Proc. of the Intl. Conf. on Fuzzy Systems*, vol. 3, pages 1809–1814, 1994.
11. L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, Part I. *Information Science*, 8:199–249, 1975.