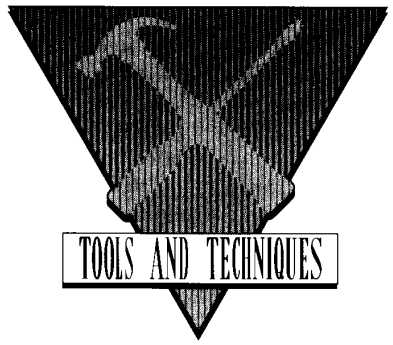


Implementing Fuzzy Rule-Based Systems on Silicon Chips

Meng-Hiot Lim, Nanyang Technological Institute
Yoshiyasu Takefuji, Case Western Reserve University

Traditionally, digital computers have been used to handle repetitive and numerically intensive tasks. But AI has created computers and software that are superior in number crunching and efficient in symbolic manipulations. Today, we associate the term AI with virtually every facet of real-world problem solving in which computers play a dominant role.

Careful analysis reveals that most so-called "intelligent" machines or programs have one common trait — they solve well-defined problems that usually can be handled with reasoning techniques based on dual logic. While this may prove adequate for many applications, a class of problems exists that does not lend itself to rigorous precision (a prerequisite for true-or-false reasoning). These problems are humanistic or ill-structured in nature; they represent problem domains that the human mind is more capable of handling. We are no longer dealing with true or false, but with the range *between* true and false. From such realizations, fuzzy mathematics emerged.

When first introduced, fuzzy mathematics was received with skepticism by experts accustomed to the rigorous precision for which classical systems are known. Despite their hesitation in adjusting to aspects of vagueness and imprecision, fuzzy logic has been successfully established as an alternative approach to reasoning in the AI community. In place of the rigorous precision for which classical systems normally strive, fuzzy reasoning is suitable for solving problems that involve vaguely defined entities. This vagueness is evident in our everyday use of such terms as "young," "tall," and "smart." Fuzzy sets can represent these vague concepts.

Over the past decade, fuzzy-set theory has been applied to solve real-world problems in such areas as control, operations research, pattern and speech recognition, expert systems, and linguistics; we find general surveys of these widely ranging applications in Dubois and Prade,[1] Negoita,[2] and Zadeh.[3] While the success of these applications proves the viability of fuzzy-set theory in AI, the

0885/9000/90/0200-0031 $1.00 © 1990 IEEE

impact has not been as great as when expert systems gained attention in the role of humanistic problem solvers — a lessened impact, perhaps attributable to the lack of tools for capitalizing on fuzzy thinking's benefits. This should motivate a software environment conducive to developing fuzzy reasoning systems.

We will address the implementation of a fuzzy simulator (FSIM) and architectures for a general-purpose VLSI fuzzy-inference processor. The FSIM tool aids in the rapid prototyping of a fuzzy production system (FPS), and represents a convenient transitional medium towards the implementation of an FPS on silicon.

Before any attempt is made to develop inference-processing hardware for approximate reasoning, it is reasonable to assume that at least a prototype has been developed and verified to function according to specifications. For rule-based systems, the biggest problem is integrating a rule set to form a coherent knowledge base. This includes the subjective definition of fuzzy terms used in these rules, and the use of proper inference mechanisms for deriving conclusions. We will show how to simplify these tedious and time-consuming tasks significantly, using a tool to aid FPS development by simulating the reasoning process. The simulator enables users to explore the various inference mechanisms proposed by fuzzy-logic researchers.

Essentially, the two areas where special-purpose hardware can be useful are real-time computations and compact (portable) computing machines. Real-time AI systems are not new. Many applications exist in which decisions must be made in real time — closed-loop control systems, for example, where considerable data from sensors must be processed before a decision can be made. An expensive (and inefficient) alternative to real-time computations would be to dedicate a powerful host to the problem. Another possibility would be special add-on hardware to accelerate CPU-intensive routines typical in problems that can be solved algorithmically.

Application-specific hardware is also used to make products small and portable — one reason behind application-specific integrated circuit (ASIC) popularity. In the near future, hand-held expert systems may be as common as pocket calculators. We hope this article represents a positive step in that direction.

Togai and Watanabe developed the first inference engine based on fuzzy logic — their CMOS implementation of a VLSI inference engine for approximate reasoning.[4] They interpreted the conditional statement "if X is A then Y is B" to be the relation $R = A \times B$, which Mamdani and Assilian used successfully in rule-based control systems.[5] However, such an interpretation is not always appropriate for modeling human reasoning. The literature presents various interpretations of the implicational relation.[6]

Togai and Watanabe's inference chip implementation shows that hardware complexity increases greatly if another implicational relation is chosen. We will discuss an alternative architecture for realizing the fuzzy-inference engine in hardware — a processor architecture conceived with the emphasis on generality (a particularly useful factor for accommodating the many possible inference methods in fuzzy reasoning).

We will present a brief theoretical review behind fuzzy reasoning, introduce the FSIM, and discuss FPS development using the FSIM. In describing our approach towards realizing expert systems on silicon chips, we will provide an overall picture of the various stages involved in developing a fuzzy-inference processor.

Next, we will outline the general architecture of a VLSI inference processor for FPSs. To further illustrate the development of a fuzzy inference processor, we will describe an FPS example from conceptualization to implementation on silicon chips.

Approximate reasoning

Fuzzy logic allows premises and conclusions to be fuzzy propositions. We can express truth values of propositions as linguistic variables with varying degrees of truth (including "true," "more or less true," and "very true"). If expert system input is in linguistic form, translation transforms the input into fuzzy sets. We achieve the reverse process, retranslation — that is, from fuzzy set to linguistic form — by means of linguistic approximation. Reasoning usually involves the handling of premises represented as fuzzy sets, and normally produces fuzzy sets as conclusions. To make sense of a deduced conclusion, using linguistic approximation during retranslation produces a final conclusion that is approximate rather than exact.

We will reason with premises or propositions of the form "X is A" (for example, "demand is high"). Four types of translation rules exist for approximate reasoning. Translation rules are a rule set used to associate fuzzy premises with their corresponding possibility distribution functions. The four types of translation rules are used for

(1) Modification,
(2) Composition,
(3) Quantification, and
(4) Qualification.

Particularly relevant in our discussions will be translation rules for modification and composition. Zadeh provides an exposition to the theory behind quantification and qualification rules.[7]

Modification. Let X be a variable and A be a fuzzy subset, both in the domain U. Consider the atomic proposition P = "X is A": The translation of P can be written as

$$P \Rightarrow \pi_X = \mu_A(u) \quad (1)$$

where u is a generic element of the variable X, and π_X is the possibility distribution function characterized by μ_A (the membership function of A). The translation of a modified proposition P* = "X is mA" where m is a fuzzy modifier — or linguistic hedges such as "not," "very," and "more or less" — is given by

$$P^* \Rightarrow \pi_X = \mu_{A^*}(u) \quad (2)$$

where $A^* = mA$. However, suppose that m = not. If so, then

$$A^* = \int_U (1 - \mu_A(u))/u \quad (3)$$

where $\int$ denotes the union of fuzzy singletons. Some examples of modifiers and mathematical definitions commonly used are

$$m = \text{"very,"} \qquad A^* = \int_U (\mu_A(u))^2/u \quad (4)$$

$$m = \text{"more or less,"} \qquad A^* = \int_U \sqrt{(\mu_A(u))}/u \quad (5)$$

$$m = \text{"plus,"} \qquad A^* = \int_U (\mu_A(u))^{1.25}/u \quad (6)$$

Strictly speaking, we should view such mathematical generalizations of linguistic hedges only as standard default definitions, if no absolute definitions are supplied.

Composition. Translation rules pertaining to composition enable the translation of proposition R, which is a composite of propositions P and Q. Basically, three modes of composition are commonly employed — conjunction, disjunction, and conditional composition. Rather than providing absolute definitions of conjunctive and disjunctive operators, we will describe them mathematically as follows:

conjunctive:
$$\forall\, x_1, x_2 \in [0,1], \qquad C(x_1, x_2) \le \min(x_1, x_2) \quad (7)$$

disjunctive:
$$\forall\, x_1, x_2 \in [0,1], \qquad D(x_1, x_2) \ge \max(x_1, x_2) \quad (8)$$

Conditional composition, more commonly known as implication, is important when representing domain knowledge in rule-based systems. The conditional proposition $R \equiv$ "if P then Q" will be symbolically expressed as

$$R \equiv P \rightarrow Q \quad (9)$$

and is read as "A implies B" (we will present mathematical interpretations of the $\rightarrow$ operator in the next section, and will describe the *modus ponens* rule of inference in the fuzzy sense).

Generalized *modus ponens*. The basic constructs of fuzzy reasoning are propositions or premises that have implied fuzzy meaning. A fuzzy proposition is an assertion about the value of a fuzzy variable. In general, each fuzzy premise contains one or more clauses, and each clause is an atomic proposition with an attribute, an object, and a value. The object is usually associated with an implied attribute and the value is represented as a fuzzy subset. For example, "John is tall" is an assertion regarding the height (attribute) of John (object) as being tall (value = membership function *tall*). Similarly, "the stock market is bullish" can be decomposed into "Wall Street market indicator" (attribute), "stock market" (object), and "bullish" (value) — realizing that more than one way usually exists for interpreting each clause.

By applying the rules of inference in fuzzy logic, we can deduce a proposition (conclusion) from a set of known premises. A special case of the rule of inference exists when

$$\begin{aligned} &P \equiv \text{"if } X \text{ is } A \text{ then } Y \text{ is } B\text{"} \\ &Q \equiv \text{"}X \text{ is } A'\text{"} \\ &R \equiv \text{"}Y \text{ is } B'\text{"} \end{aligned} \quad (10)$$

This is known as compositional *modus ponens*, and can be expressed as

$$B' = A' \circ (A \rightarrow B) \quad (11)$$

where $A \rightarrow B$ is an implicational relation read as "A implies B" and "$\circ$" is a fuzzy max-* composition operator (read as max-star composition, where "*" is an operator to be defined). We can view the above as a generalization of *modus ponens* that reduces to classical *modus ponens* when $Q \equiv$ "X is A" and $R \equiv$ "Y is B" (A, A', B, and B' are fuzzy concepts while X and Y are variables representing objects).

In knowledge engineering terminology, we commonly refer to a rule of the form "if <antecedent> then <consequent>" as a production rule. For FPSs, a chain of rules (also commonly referred to as fuzzy algorithms when applied to a known fact or observation) will deduce the conclusion after each pass through the rule set. Typically, fuzzy productions are of the form

$$\begin{aligned} &\text{Rule 1:} \quad \text{if } X \text{ is } A_1 \text{ then } Y \text{ is } B_1 \\ &\text{Rule 2:} \quad \text{else if } X \text{ is } A_2 \text{ then } Y \text{ is } B_2 \\ &\vdots \qquad\qquad \vdots \\ &\text{Rule n:} \quad \text{else if } X \text{ is } A_n \text{ then } Y \text{ is } B_n \end{aligned} \quad (12)$$

In contrast to a classical production system, all FPS rules are considered to be fired (but with different strength). Of course, rules that fire strongly will contribute significantly to the final conclusion. Since all rules are said to contribute to the final conclusion, it makes sense

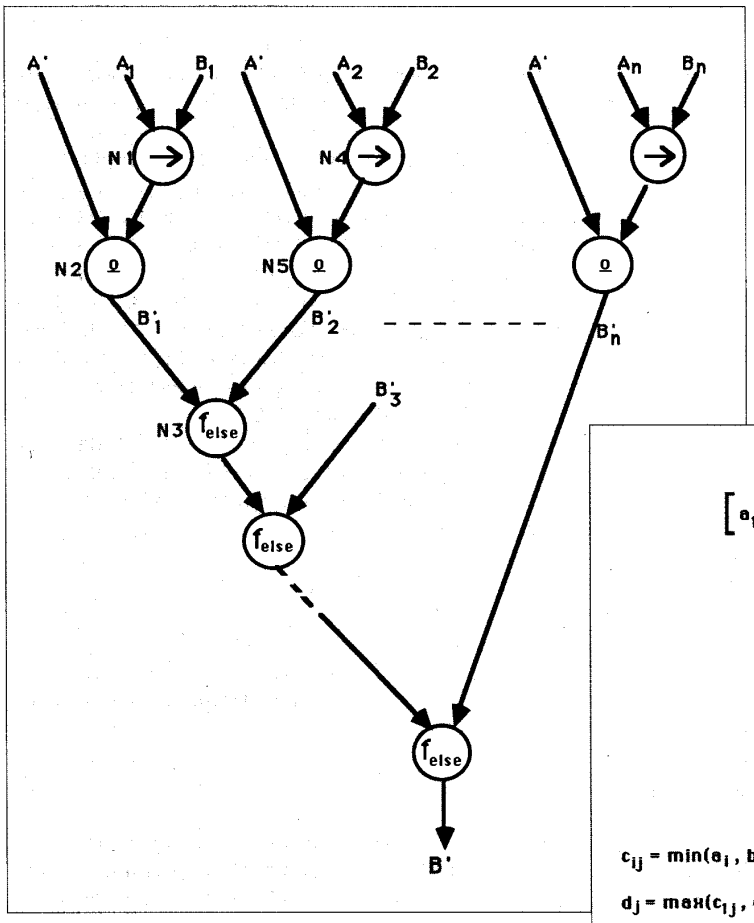


Figure 1. The decision tree of a fuzzy production system.

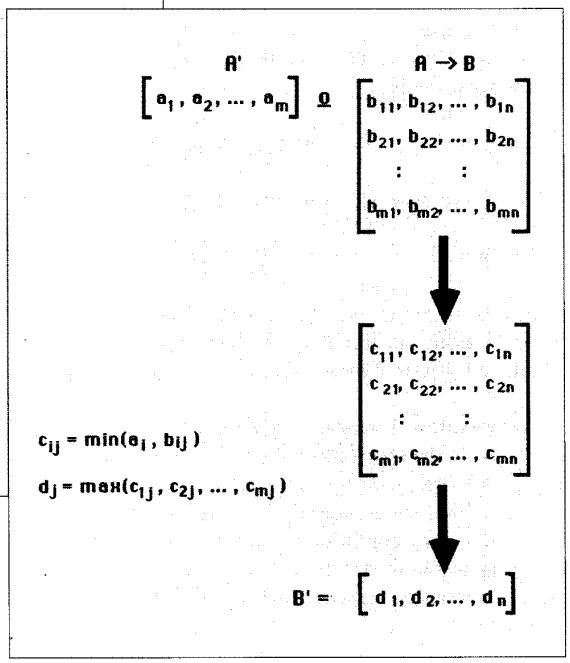


Figure 2. Max-min composition.

to include an additional operator that will aggregate conclusions deduced from each rule. To this end, the f_{else} operator is introduced — which will be referred to as the production link and is significant when determining how concluded values are propagated after each rule is fired.

Given the FPS of Equation **(12)** above, we can construct a decision tree of nodes and branches as shown in Figure 1. To achieve the conclusion B', given the fuzzy input pattern A', all nodes are triggered in a top-down and left-right manner. Each node is triggered when the values of branches (fuzzy premises) are known. In Figure 1, the N1 node is first triggered, followed by N2. At this point, the B_1 branch would have been known. Node N3 cannot be triggered yet, since the B_2 branch is still unknown. However, conditions are right for triggering N4 and (subsequently) N5. Once the value for the B_2 branch is known, N3 will then be triggered. This process repeats until all nodes have been triggered, thus producing the FPS conclusion B'.

Clearly, three different operators have been identified: implication, max-* composition, and the production link f_{else}. Hereafter, unless otherwise stated, we assume these operations to be generic in nature (that is, representing any of the different operations possible in the context of approximate reasoning). In the discussions that follow, we will describe the various possible interpretations of these operators.

Fuzzy implication. Many studies have been done to obtain a fuzzy operation for $\rightarrow$ of Equation **(9)** that fits the context of the fuzzy conditional proposition. For a qualitative comparison on the different implications, see our references.[6,8-10]

The $\rightarrow$ operator can be any of the 15 fuzzy implications listed by Mizumoto and Zimmermann.[6] Let A and B be fuzzy subsets in universes U and V, respectively, as follows:

$$A = \int_U \mu_A(u)/u, \qquad B = \int_V \mu_B(v)/v \tag{13}$$

and for simplicity, let

$$a = \mu_A(u), \qquad b = \mu_B(v) \tag{14}$$

Let $\cap$, $\cup$, $\neg$, $\times$, and $\oplus$ denote (respectively) the intersection, union, complement, Cartesian product, and bounded sum operation. For example, consider $A = 0.5/1 + 0.7/2 + 1.0/3$ and $B = 0.4/3 + 1.0/4 + 0.4/5$, where "+" signifies union rather than arithmetic sum. If the relation R_m is chosen for the conditional proposition "if X is A then Y is B," then

$$\begin{aligned} R_m &= (A \times B) \cup (\neg A \times V) \\ &= \int_{U \times V}((a \wedge b) \vee (1 - a))/(u, v) \\ &= 0.5/(1,3) + 0.5/(1,4) + 0.5/(1,5) \\ &\quad + 0.4/(2,3) + 0.7/(2,4) \\ &\quad + 0.4/(2,5) + 0.4/(3,3) + 1.0/(3,4) + 0.4/(3,5) \end{aligned} \tag{15}$$

Fuzzy max- composition.* The fuzzy conditional statement "if X is A then Y is B" is said to be a causal link from X to Y. In practice, all rules in a fuzzy production system attempt to describe the causal link between X and Y as thoroughly as possible.[11] The atomic propositions "X is A" and "Y is B" can be expressed as

$$\pi_x = \mu_A(u) \text{ and } \pi_y = \mu_B(v) \tag{16}$$

Here, π_x and π_y are possibility distribution functions restricting the possible values of X and Y, respectively.

Given the proposition "X is A'" and the reference proposition "if X is A then Y is B," we can deduce the consequent "Y is B' " by means of the compositional rule of inference described in Equation **(11)**. The max-* composition operator "$\circ$" in Equation **(11)** is also commonly referred to as a detachment operator. The most frequently used max-* compositional operator in practical applications of fuzzy logic is Zadeh's max-min composition.[12] Figure 2 illustrates the procedure for computing B' by max-min composition.

Other possible compositional operators have also been suggested; for example, max-Θ composition and max-Λ composition. The operators Θ and Λ — introduced by Dubois and Prade[1] — are referred to as bounded-product and drastic-product, respectively, in Mizumoto.[6] They are defined as follows:

$$\forall\, x_1, x_2 \in [0,1], \qquad x_1 \,\Theta\, x_2 = 0 \vee (x_1 + x_2 - 1) \tag{17}$$

and

$$\forall\, x_1, x_2 \in [0,1], \qquad x_1 \,\Lambda\, x_2 = \begin{cases} x_1 & x_2 = 1 \\ x_2 & x_1 = 1 \\ 0 & x_1, x_2 < 1 \end{cases} \tag{18}$$

The production link. Production system rules are chained together in either a causal or noncausal manner. Causally chained rules usually signify the dependence of a rule on a previously fired rule. In contrast, two rules are said to be noncausal if the firing of one rule is not affected by the other; that is, noncausal rules are independent of one another and each fires autonomously. In the FSIM, the rules of Equation **(12)** can be considered as knowledge chunks with the distinction that they all have the common right-hand side (RHS) object-attribute Y. Rules within a knowledge chunk must be noncausal. On the other hand, two knowledge chunks can be causal. The order in which causal chunks of knowledge are specified is important; that is, rules for deducing attribute values that other knowledge chunks will use must be specified first.

Let's examine three possibilities of f_{else} fuzzy operation to model the "else" connective in the FPS of Equation **(12)** (the possibilities of suitable aggregation operators are certainly not limited to the three described below; other suitable operators can be introduced to better model human decision making, as Zimmermann and Zysno have demonstrated):[13,14]

(1) *The or-link* (f_{or}) — In classical production systems, where premises are not fuzzy, a rule is said to be fired if an input pattern exactly matches the pattern on a rule's left-hand side (LHS). In contrast, all rules are considered to be fired (but with different strength) in FPSs. As such, the final deduced conclusion is said to be a contribution of all these rules. The simplest mathematical operation for realizing the "Or" aggregation is by means of the max operation. The *or-link* tends to have a monotonic effect on reasoning. In a different perspective, we can consider rules that are "Or" aggregated as generalized monotonic reasoning, since all deduced conclusions hold true (but to various extents) throughout reasoning. A concluded value is either true or false for monotonicity in classical knowledge-based systems. Usually, premises concluded from weakly fired rules tend to have little or no effect on the end result. Many authors (Mamdani and Assilian, for example[5]) have used f_{or} as the production link in their applications.

(2) *The and-link* (f_{and}) — Instead of the *or-link*, conclusions from all FPS rules can be aggregated using the *and-link*. The simplest mathematical operation for realizing the "And" aggregation is by means of the min operation. Dubois and Prade[11] suggest using the fuzzy min operation to aggregate the result in an FPS, particularly when $\rightarrow$ is derived from Lukasiewicz's implicational logic (as explained by Rescher),[15] which corresponds to the relation of Equation **(15)**. Another possibility for using f_{and} (or f_{or}) would be to divide a single rule into multiple rules — particularly necessary when a rule's antecedent contains too many preconditions or clauses to be specified in a single rule.

(3) *Truth-qualification* (f_τ) — A deduced conclusion can have an associated truth or confidence level, if

necessary. Referring to equation **(11)**, the truth level of the deduced premise "Y is B'" can be defined as the ratio $\tau_{B'}$, expressed as follows:

$$\tau_{B'} = \Sigma\text{-count}(B')/\Sigma\text{-count}(B) \quad \textbf{(19)}$$

where

$$\Sigma\text{-count}(B') = \Sigma\mu_{B'}(v), \qquad \forall\, v \in V \quad \textbf{(20)}$$

and

$$\Sigma\text{-count}(B) = \Sigma\mu_{B}(v), \qquad \forall\, v \in V \quad \textbf{(21)}$$

A necessary precondition for Equation **(19)** to apply is that the conclusion B' must be a subset of B. Let τ_i denote the value of $\tau_{B'}$, due to the ith rule in Equation **(12)**. Then, in general, the greatest value of τ denoted by τ' can be written as

$$\tau' = \vee\tau_i, \qquad \text{for } i = 1,\ldots,n \quad \textbf{(22)}$$

where $\vee$ denotes the max operation. If $\tau' = \tau_k$, the final conclusion B′ will be equal to B_k — meaning that deduced attribute's value is the translation of the premise with the highest truth level. This operation resembles classical rule-based systems in which only one rule contributes to the final conclusion.

Briefly stated, the selected f_{else} operation depends primarily on the context in which rules are written. Other possible connectives can be derived to suit the knowledge source's semantics requirement. To suit the problem domain, however, it is sometimes necessary to provide a different interpretation of the f_{else} operator. While this may add to inference procedure complexity, it could prove indispensable for applications in which human decision making capabilities must be better modeled.

Simulation with the FSIM

Reasoning with concepts involving fuzzy sets requires more sophisticated matching and rule-firing mechanisms than those offered by conventional two-valued propositional logic. We can broadly classify steps involved in FSIM simulation as follows:

(1) Specify fuzzy production rules,
(2) Define fuzzy terms as possibility distributions,
(3) Parse the rules,
(4) Convert rules into symbolic representations,
(5) Choose the appropriate inference mechanism, and
(6) Simulate with test cases; if unsatisfactory, go back to steps (1) or (2).

A brief outline follows of the FSIM's role in the development of fuzzy production systems. Lim provides further details.[16]

The input syntax. Input to the FSIM is specified as a chain of rules (rule1, else rule2, else ..., else ruleN) in which each rule contains the LHS clause(s) referred to as antecedent and the RHS clause for the consequent. Each rule's LHS comprises one or more clauses, whereas the RHS supports one clause consequent. Limiting the RHS to a single clause does not restrict the system in any way, since a consequent with multiple clauses can usually be decomposed into separate knowledge chunks. Multiple clauses on the LHS are linked together by logical connectives (LC ≡ {and, or}). In general, each rule is represented as follows:

```
If
                <antecedent_clause_1>
LC              <antecedent_clause_2>
  :                       :
  :                       :
LC              <antecedent_clause_n>
Then
                <consequent>
```

All clauses are enclosed within < and > symbols, and each clause is of the form "$X is attribute." The $ symbol is used to denote the presence of an object that is analogous to a variable in mathematical equations. Hence, we can view tokens preceded by a $ symbol as fuzzy variables. Reserved words with special meanings cannot be used to name variables or attributes; "if," "then," "else," "and," "or" — and other predefined fuzzy modifiers (linguistic hedges) including "not," "very," and "most." Users can override built-in definitions and define new fuzzy modifiers and connectives by preceding the intended word with a ~ symbol. For example, "~very" and "~anyword" will cause the program to search for functions in the FSIM's user-defined library. Each clause's attribute can be complex; for example,

P = "not very young and not ~too old"

The convention for specifying compound attributes is "operator (operand1, operand2)." A proposition of the form "the man is P" would be specified as "$the_man is and(not very young, not ~too old)."

We developed the FSIM's parser with help from YACC (yet another compiler compiler), a program that generates parsers. YACC is available as part of the Unix operating system's software distribution. The FSIM's parser functions are threefold:

(1) To detect syntactical errors in input;
(2) To extract the list of token names that are variables or attributes that users must define before inference can proceed; and
(3) To convert high-level user representation of fuzzy production rules into the FSIM symbolic format used to drive inferencing.

To present the FSIM's overall syntax structure, Table 1 shows the YACC's input description. Successful rule parsing will produce three disk files — a symbolic representation of the rules, a list of fuzzy attributes, and a list of input variables.

Table 1. Input text for YACC.

```
                    /* Production rules parser */
%token
IF IS THEN ELSE F_MODIFIER F_VAR F_LABEL LOGICAL
%{
                    extern    main(), yylex(), yyerror();
%}
%start  rule_base
%%
rule_base           :
                    |         rule
                    |         rule ELSE rule
                    ;
rule                :         IF antecedent THEN action
                    ;
antecedent          :         clause
                    |         antecedent LOGICAL clause
                    ;
action              :         clause
                    ;
clause              :         '<' F_VAR IS attribute '>'
                    ;
attribute           :         argument
                    |         LOGICAL '(' attribute ',' attribute ')'
                    ;
argument            :         F_LABEL
                    |         LOGICAL '(' argument ',' argument ')'
                    |         F_MODIFIER argument
                    ;
%%
```

Symbolic representation. In high-level programming languages, we use compilers to convert high-level code to lower level representations (and, eventually, into machine executable codes). Symbolic representation of fuzzy production rules in the FSIM is analogous to a compiled program's machine code; that is, the symbolic form is an FSIM-executable representation. Actually, it describes an FPS's decision tree structure. We can consider each pass through the FPS as a top-down search of the decision tree.

The decision tree's nodes are associated to fuzzy operations, and the branches to fuzzy sets are characterized by the grade-of-membership functions. Two types of branch exist — terminals and nonterminals. Each branch is directed either inward or outward with respect to a node. Except for variables whose values result from inferencing, terminal branches are subjectively predefined object classes characterized by their grade-of-membership functions. Nonterminals are intermediate results during inference.

Like branches, there are also two types of nodes — uninodes and binodes. Uninodes have only two branches associated with them, one going inward and the other outward. Typically, uninodes represent operations caused by the use of fuzzy modifiers such as "not," "very," and "most" in the rules. Binodes have three branches associated with them. Each binode has two branches going inward and one outward. A node is said to be triggered if the value(s) of the branch(es) going inward is known.

The user interface. Prototypical systems are subject to constant modification during development. Rules are changed until a coherent knowledge base is achieved. A coherent knowledge base consists of a rule set with no two (or more) contradictory rules. Incorrect or ambiguous results can usually be remedied by changing rules or adding new rules. Frequently, the system dictionary must be changed to further refine the fuzzy production system's overall performance. Adjusting the knowledge base is called "tuning." We implemented the features to facilitate tuning in the FSIM's simulation environment, using the C-Language built-in utility.

Users must be able to monitor intermediate results selectively during inferencing. By tracing through intermediate inferencing stages, users can pinpoint weak or contradictory rules. The FSIM provides a graphical display to help users perform such tracing. By selectively stopping inference at various stages, users can examine the current memory state by specifying tokens associated with a fuzzy subset that will be plotted on the graphical display window. This feature aids in debugging the knowledge base and is especially useful during FPS development.

The natural language interface module, an additional user interface component, is helpful when developing FPSs. Since fuzzy reasoning involves using linguistic labels, it is generally more convenient for users to specify input in linguistic form. Linguistic input must then be converted to equivalent fuzzy sets, based on a set of predefined fuzzy terms. The FSIM incorporates such a capability, as well as the capability to accept fuzzy sets as input. The latter is necessary because in some applications — due to the nature of input sources (sensors, gauges, and meters, for example) — it is more convenient for input to be in the form of fuzzy sets. This is especially true in many rule-based control systems.

Conclusions deduced from fuzzy systems are usually formed as fuzzy sets. To assess system outcome effectively, resultant fuzzy sets must be translated into semantically equivalent language expressions comprehensible to humans. This is known as linguistic approximation, which various authors (Schmucker,[17] for example, and Degani and Bortolan[18]) have addressed both as a general problem and in the context of particular applications.

Eshragh and Mamdani,[19] and Wenstop[20] have discussed the more general problem, in which only one fuzzy subset will be translated into words. At present, no general linguistic-approximation routine has been implemented for the FSIM.

From FSIM to silicon chips

When developing a reasoning system, one of the main tasks involves formulating the knowledge base itself. Once knowledge has been captured, the system usually undergoes stages of refinement and modification until a workable system is achieved. When the system functions consistently as desired, its knowledge base becomes a static data structure unless the system has learning capabilities. The knowledge base remains unchanged until the system undergoes further modifications (usually to improve system performance and reliability as more experience and understanding are gained regarding the system's reasoning behavior under normal operations). In fact, what we have stated so far represents the prime motivation for a knowledge-based approach to solving domain-specific problems in the first place. By separating domain knowledge, we can easily upgrade sophisticated programs by independently modifying the knowledge base.

Since knowledge-based system performance expectations in certain applications have begun to exceed the limit achievable by programs running on general-purpose hosts, researchers are looking into parallel computing architecture and special-purpose hardware to address the problem. The latter has been a popular approach, as the recent surge in the ASIC market indicates. Two approaches exist for developing application-specific hardware — using customized ASIC chips, or using standard off-the-shelf chips. When faced with demanding performance expectations, the second option may not be feasible; custom VLSI chip performance is far better than that of standard off-the-shelf parts, since a circuit board's electrical parameters differ greatly from those of a silicon die. Furthermore, from the standpoint of development time, the prototyping of VLSI chips has reached a stage comparable to using standard off-the-shelf chips — due mainly to advanced state-of-the-art semiconductor processing technology and efficient CAD tools for rapid chip prototyping.

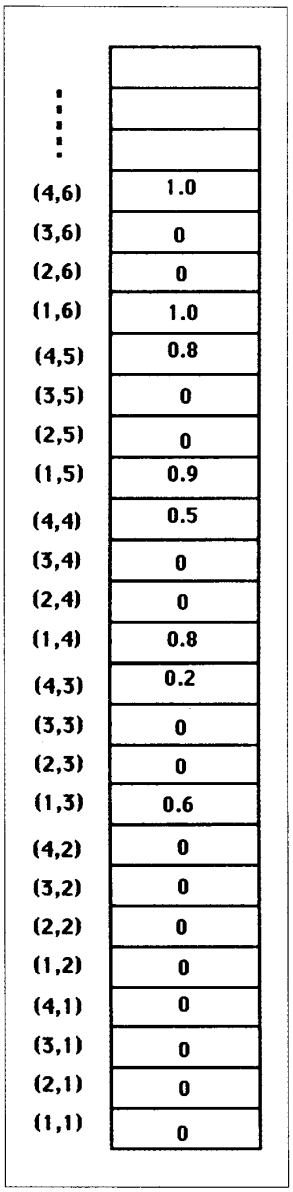


Figure 3. The implicational relation matrix in a memory element.

Knowledge representation. Assuming an application requires that an FPS be implemented in VLSI circuitry — and bearing in mind that we should avoid secondary memory storage (including hard disks) for performance considerations — how should we represent the knowledge base in a format useable by a fuzzy-inference processor?

Let's examine the methodology involved in mapping fuzzy production rules into binary matrices. The context in which we use the term "knowledge representation" refers to the method of representing the knowledge base in memory elements — random-access memory (RAM), for example, or read-only memory (ROM).

An obvious hardware equivalent of the fuzzy conditional proposition "if X is A, then Y is B" will consist of memory storage for fuzzy concepts A and B, and a hardware block to compute the implicational relation matrix. In Togai and Watanabe's implementation of a VLSI fuzzy inference engine, the following implicational relation was assumed:[4]

$$R_{A \rightarrow B} = \min(\mu_A(u), \mu_B(v))/(u, v) \quad (23)$$

This instance computes the implicational relation by means of a simple *min* hardware module. However, as described earlier, such an implicational relation is not always appropriate for modeling human-like reasoning. From a hardware implementation viewpoint, a problem arises because $\rightarrow$ can be any of the many possible implicational relations described. In other words, we must design special hardware for each of the 15 relations outlined by Mizumoto and Zimmermann.[6] While feasible for certain implicational relations, it can be difficult and expensive to design special hardware for some relations — especially those involving multiplication and division. Moreover, as implicational relation block complexity increases, the hardware system's overall control strategy becomes more complicated.

An alternative approach would store the precomputed "$A \rightarrow B$" relation matrix directly into the processor's main memory, greatly reducing hardware complexity and simplifying the inference engine's control strategy. For example, let A and B be fuzzy subsets of the universe $U = \{1, 2, 3, 4\}$ and $V = \{1, 2, 3, 4, 5, 6\}$, respectively.

If

$$A = 0.5/1 + 1.0/4 \quad (24)$$

$$B = 0.2/3 + 0.5/4 + 0.8/5 + 1.0/6 \quad (25)$$

and we take $R_{A \to B}$ to be the relation of equation (23), then

$$R_{A \to B} = 0.2/(1,3) + 0.2/(4,3) + 0.5/(1,4) + 0.5/(4,4) + 0.5/(1,5) + 0.8/(4,5) + 0.5/(1,6) + 1.0/(4,6) \quad (26)$$

Similarly, if the R^* relation is chosen,

$$R^* = (A \times V) * \Rightarrow (U \times B) = \int_{U \times V} (1 - a + ab)/(u, v) \quad (27)$$

where a and b are as defined in equation (14), then

$$R_{A \to B} = 0.6/(1,3) + 0.2/(4,3) + 0.8/(1,4) + 0.5/(4,4) + 0.9/(1,5) + 0.8/(4,5) + 1.0/(1,6) + 1.0/(4,6) \quad (28)$$

Figure 3 shows how the $R_{A \to B}$ matrix of equation (28) can be stored directly into memory as knowledge to be used for reasoning. In equations (23) and (27), (u,v) are mapped onto hardware as the memory address of implicational matrix values in memory storage elements.

The overall approach. The three major stages of FPS implementation are problem conceptualization, working prototype, and inference processing hardware. Figure 4 overviews the development of an FPS on silicon; the prototypical system is developed by means of the FSIM. Once the system's functionality has been verified, we can convert the knowledge base into relational matrices suitable for binary representation in the hardware engine's main memory. The "dump" command accommodates this feature in the FSIM, creating a disk file with arrays of values corresponding to implicational relation matrices. For each relation matrix, a two-dimensional array of grade-of-membership values is created. These values are then loaded into the inference processor's main memory from a general-purpose host machine. For example, on Sun workstations, we can program the inference processor's main memory by configuring it as a multibus board inserted into the card cage. We can then use software routines to access specific board memory locations for read or write operations. Borriello et al. provide an example of user-level C routines for this purpose.[21]

Clearly, the host computer is crucial during developmental stages only and does not remain a permanent fixture of the hardware engine. Once the knowledge base has been loaded, the hardware engine can function as an autonomous reasoning system provided that the proper peripherals for interfacing with external I/O devices are available. If we do not expect the knowledge base to change, we can use a more permanent form of memory storage. For example, we can use ROM to encode knowledge for the processor once the knowledge base is fixed. Even then, we can accommodate further knowledge base changes by replacing the ROM chip. Besides making the reasoning engine small and portable, such architectural flexibility makes it suitable for certain applications in which the expert knowledge of different human experts can be made easily available in the form of encoded memory chips.

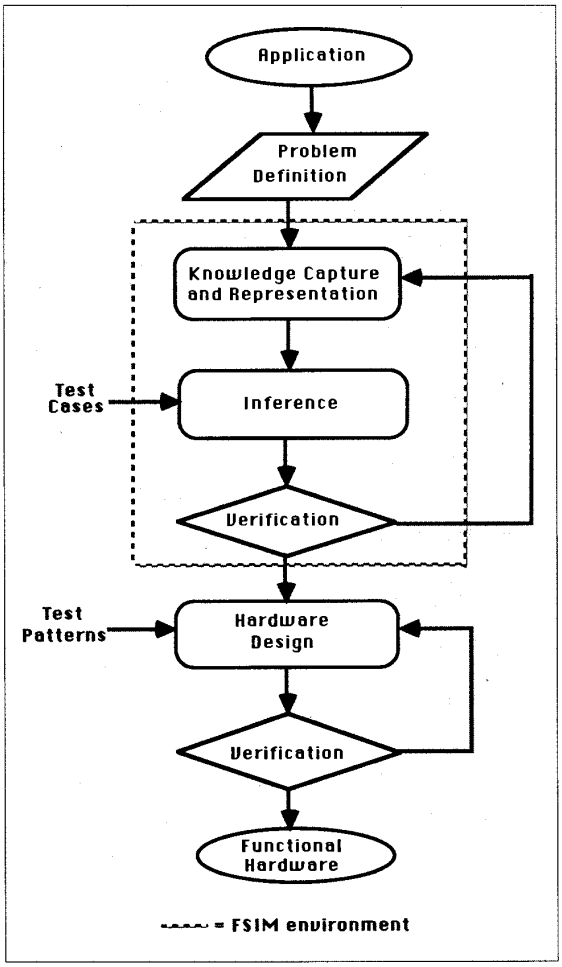


Figure 4. Stages of FPS development.

VLSI inference processor architecture

We can separate inference processor design into four basic functional blocks — the knowledge base (main memory), the control engine, the composition engine, and the aggregation engine. Figure 5 shows the overall architecture. In addition to these four functional blocks are two register arrays. Register array A stores external fuzzy input values. Register array B stores intermediate results of computation.

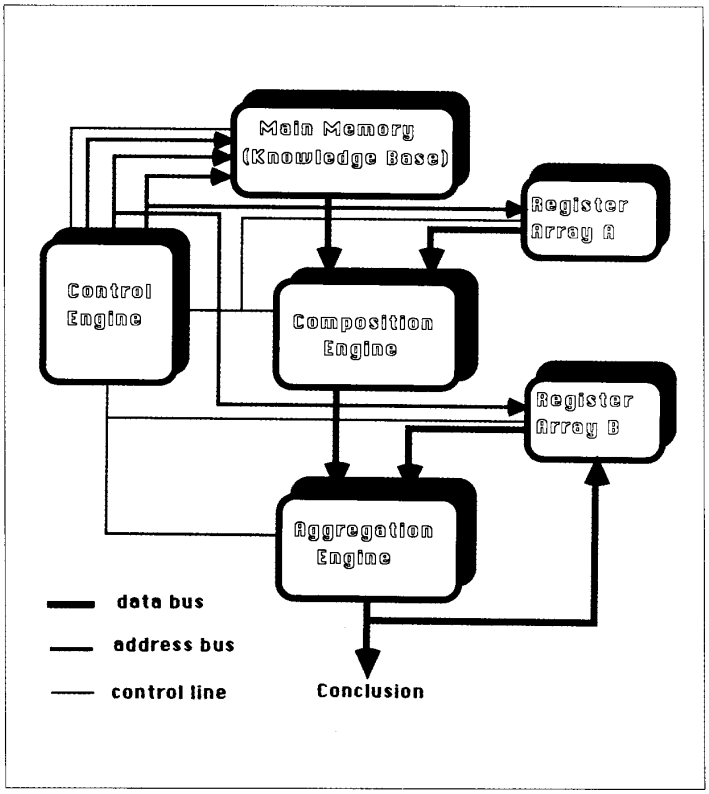


Figure 5. The overall system architecture of a fuzzy inference processor.

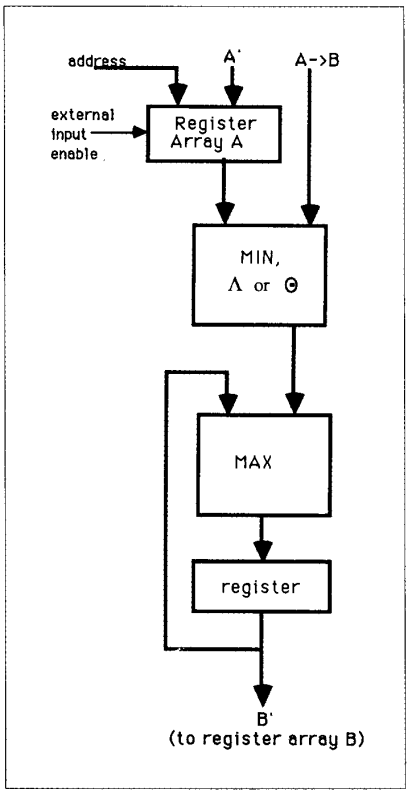


Figure 6. The composition engine's data path.

The reasoning engine's hardware architecture is suitable for use in a master/slave configuration with an external device including a host or other application-dependent I/O devices.

The knowledge base (main memory). For the architecture being described, the knowledge base resides in main memory to minimize I/O overhead. A general-purpose host machine computes fuzzy implicational relation matrices, which are then loaded into the inference processor's main memory. This minimizes hardware complexity by eliminating the on-chip computation required for implicational relation matrices. In addition, processor flexibility is evident since the hardware is independent of the implicational relation chosen for the reasoning system. Such generality allows more complicated inference mechanisms to be used without incurring additional expense for hardware.

The control engine. One of the controller's main functions is to generate the address for accessing the knowledge base in main memory. In addition, it synchronizes the processor's overall operation. The key behind the system's basic control strategy lies in the control engine's three count status blocks: U-count, V-count, and rule-count. Rule-count is incremented after each rule has been processed. V-count is incremented after each column of the relation matrix has been processed (see Figure 2), and resets after each matrix has been totally processed. U-count addresses input from register array A. It resets when all elements in the A' matrix and a column of the implication matrix in Figure 2 have been processed. As we will examine later, the control engine's main components are synchronous counters.

The controller also generates signals to control proper operational sequencing among modules. Each time a column of elements in the relation matrix has been processed, the aggregation engine is activated. When activated, the aggregation engine updates the contents of register array B (which contains the concluded fuzzy attribute's current values). After each complete pass through the FPS, the system can use a control flag to signal any external device that data in register array B is currently valid (that is, that the final concluded value has been attained). Register array A stores externally loaded fuzzy-input patterns. All other operations are suspended while register array A is being loaded with external input.

The composition engine. The composition engine takes input from register array A and the main memory ($A \rightarrow B$) and produces the outcome B', which is stored in register array B. Max-min composition has been the most common detachment operator used in fuzzy-logic applications. Other detachment operators are max-Λ and max-Θ (described earlier). Figure 6 shows the data path for realizing the composition engine. Inferencing is seldom affected by the composition operator chosen. In most cases, max-min composition seems sufficient to model the composition *modus ponens* of equation **(11)**. The value d_j (see Figure 2), the maximum of all values in the jth column of the c matrix, will be stored in the register (see Figure 6). At this point, the aggregation engine updates the deduced conclusion.

Table 2. Rules for FXLoan.

```
#
#       $cscore = critical score.
#       $cratio = critical ratio.
#       $ccredit = critical credit
#
if              <$cscore is high>
        and     <$cratio is good_cr>
        and     <$ccredit is good_cc>
then            <$decision is approve>

else
if              <$cscore is low>
        and     <$cratio is bad_cr>
        or      <$ccredit is bad_cc>
then            <$decision is disapprove>
```

The aggregation engine. The production rules represented as relational matrices in memory are constrained to be of single-clause antecedent and single-clause consequent. Rules with compound antecedents (that is, multiple clauses with "and" or "or" connectives) can be decomposed into multiple rules by using the appropriate f_{else} production link. We assume that all fuzzy sets representing the antecedent are characterized by grade-of-membership values spanning the universe of discourse of the same size, which reduces the control unit's hardware complexity significantly.

We have described three different operators to interpret the "else" connective in a fuzzy production system. Except for the f_τ operation, the other two corresponding f_{else} operators can be realized easily. The basic building blocks of the aggregation engine in its simplest form consist of a min module for f_{and} and a max module for f_{or}. Other aggregation operators to model the FPS's "else" connective can be implemented, if necessary. To realize the f_τ operator, for example, the overall system will require more complicated control strategy since division of two binary numbers is involved. Accurate division is not necessary, because membership values have been "discretized" into levels; a pseudodivider can be implemented with the use of an adder, a register, a synchronous counter, and a magnitude comparator. Alternatively, a table-lookup method (via a programmable-array-logic module) can be used as a divider for two binary numbers. Division using programmable array logic requires only one clock cycle. Besides being faster, PAL can reduce the complexity of overall control strategy. Its main disadvantage is its size, which increases as "discretized" levels of membership values increase in number.

The design example

We have described an overall approach for developing FPSs. To further clarify this approach, we will examine a case study from the problem specification to the inference processor's hardware implementation. This example involves developing an expert system — which we'll call FXLoan (fuzzy X loan) — for processing bank loan applications. A nonfuzzy version of a system called XLoan has already been developed. First, we will explore the problem domain briefly. Then, we will demonstrate the inference processor's design. Step-by-step coverage of FXLoan's implementation will link all the steps and procedures outlined earlier.

This example represents a simplified version of an actual system. We will show only the portion of the system that actually reasons. A more involved FXLoan would require an elaborate user interface just for data acquisition. Practically speaking, this exercise's final outcome will be fuzzy-inference processing hardware that is programmable with domain knowledge. A promising applicational aspect of the inference processor will be its suitability for developing a small and portable (hand-held) FXLoan.

The knowledge base. Financial institutions process many loan applications every day, and interest charges represent a major source of their income. Loan officers — who assess most loan applications — gather information regarding income, credit history, age, and assets that indicate a loan applicant's creditworthiness. Loan officers determine whether or not to approve loan applications, based on collected data, preconceived guidelines, and experience; they usually arrive at their decisions via a pencil-and-paper approach.

XLoan's domain knowledge was acquired through interviews with a bank loan officer. This expert's factual and heuristic knowledge was represented as production rules. From user input — age, assets owned, credit history, employment, and so forth — XLoan concludes whether or not to approve loan applications.

FXLoan's knowledge base consists of two fuzzy production rules (see Table 2). The knowledge base contains three fuzzy input variables — $cscore, $cratio, and $ccredit. When the values of these variables are provided as input, FXLoan will deduce a conclusion that is the value of the variable $decision. XLoan contains 21 rules to account for each possible combination of the observables *critical score*, *critical ratio*, and *critical credit* — typical of exact reasoning methodology. Since

the comparable version of FXLoan requires only two rules, FXLoan approximates the heuristic approach that humans use in reasoning.

The FSIM's simulation. Once we have defined fuzzy production rules, the next step is to simulate the system's reasoning behavior, using the FSIM. First, rules are parsed to check for syntactical errors. After successful parsing, three additional files are produced. One file contains the symbolic representation, which describes the decision tree. The other two files contain listings of fuzzy labels and input variables used in the rules. Figure 7 shows the corresponding decision tree for FXLoan. Lim provides further details on the procedures involved in simulating FXLoan.[16]

The dictionary, an additional support file necessary for the FPS, contains user-supplied definitions on label meanings in

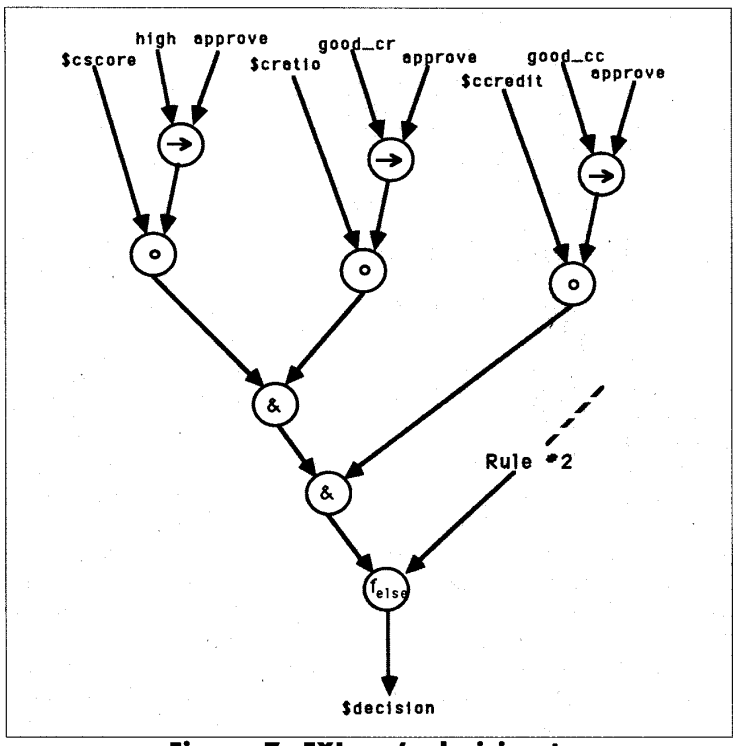


Figure 7. FXLoan's decision tree.

Table 3. A dictionary file for FXLoan.

```
# Definition of fuzzy terms used in FXLoan.
#
# Universe to define acceptability of $cscore.
# Elements:
#        150     155     160     170     175     180     185     190     195     200
~high                    11
         0.0     0.0     0.0     0.0     0.0     0.0     0.2     0.7     1.0     1.0
~low                     11
         1.0     1.0     0.8     0.5     0.2     0.0     0.0     0.0     0.0     0.0
#
# Universe to define acceptability of $ccredit.
# Elements:
#        0       1       2       3       4       5       6       7       8       9       10
~good_cc                 11
         1.0     1.0     1.0     0.7     0.3     0.0     0.0     0.0     0.0     0.0     0.0
~bad_cc                  11
         0.0     0.0     0.0     0.0     0.0     0.0     0.3     0.7     1.0     1.0     1.0
#
# Universe to define acceptability of $cratio.
# Elements:
#        0.1     0.3     0.4     0.41    0.42    0.43    0.44    0.45    0.5     0.7     1.0

~good_cr                 11
         1.0     1.0     0.7     0.3     0.0     0.0     0.0     0.0     0.0     0.0     0.0
~bad_cr                  11
         0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.3     0.7     1.0     1.0

#
# Universe to define $decision.
# Elements:
#        0       1       2       3       4       5       6       7       8       9       10
~approve                 11
         0.0     0.0     0.0     0.0     0.0     0.0     0.3     0.7     1.0     1.0     1.0
~disapprove              11
         1.0     1.0     1.0     0.7     0.3     0.0     0.0     0.0     0.0     0.0     0.0
```

Table 4. Implicational relation matrices for FXLoan.

1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	*(high -> approve)*
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	*(good_cr -> approve)*
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	*(good_cc -> approve)*
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	
1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	

terms of fuzzy sets (see Table 3). Based on the rules specified, we evaluated various inference mechanisms to determine the suitability of each inference method. By providing various test cases and observing the FSIM's deduced conclusions, we found a few fuzzy implicational relations to be suitable — R_c, R_{sg}, R_{gg}, R_{gs}, and R_{ss}.[6] The system determines each relation's suitability by assessing the conclusion deduced from test cases and then visually approximating the fuzzy set into linguistic form. Therefore, evaluating each inference method is highly subjective. Experience gained from various trials of the inference mechanisms strongly suggests that the context in which rules are specified greatly influences the choice of the implication relation.

Once we have chosen the proper inference mechanism and exercised the knowledge base enough to verify its functionality, the next step is to convert fuzzy production rules into implicational relation matrices suitable for representation in the fuzzy-inference processor's main memory. We accomplish this via the "dump" command in the FSIM, which outputs implicational matrices on a disk file. Table 4 shows the first three of the six implicational matrices for the R_{sg} relation. These matrices are then loaded into the inference processor's main memory.

Hardware implementation. FXLoan's hardware implementation (simplified in this example) is small enough that we can realize the system without much difficulty using standard off-the-shelf SSI and MSI chips. Implicitly, the ultimate target will be to realize the processor using VLSI technology for maximum performance.

As far as inference processing hardware design is concerned, the system can be treated as if it were a six-rule FPS. Table 3 defines fuzzy labels such that all universe sets contain the same number of elements. Each universe has 11 members. This simplifies the design significantly by eliminating the need for additional counters and a more complicated control strategy (if the universe of discourse were chosen to be of a different size). The grade-of-membership values are "discretized" to 11 levels (0, 0.1, ..., 1.0). Therefore, we need a 4-bit binary number (0000_2, ..., 1010_2) to represent the grade-of-membership values.

Figure 8 shows the overall system design that we can realize using standard off-the-shelf chips. We can realize register arrays for the processor by means of register file chips (74870). We chose max-min composition in this design. The shift register selects the appropriate operational mode for the magnitude comparator (0 for min and 1 for max) in the aggregation engine. After each rule has been processed, the shift register is shifted. The VALID line is for external handshaking, and is asserted when the final FPS rule is being processed. This signifies that processor output is currently valid (that is, that the final conclusion is being deduced). The INEN operation (input enable) suspends all other operations, and loads the register file with external input (X is A').

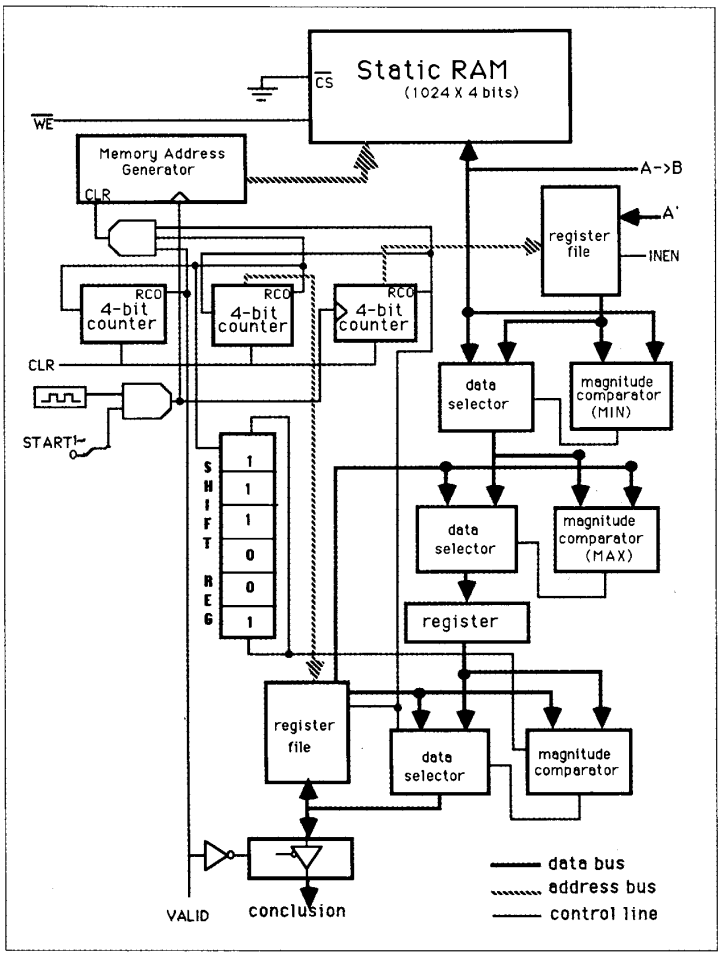


Figure 8. FXLoan's hardware design.

Memory size. Since the system contains six rules, six implicational relation matrices must be stored in main memory. Each relation matrix consists of an 11-by-11 array of membership values. Therefore, the total number of memory bits required for each rule will be $121 \times 4 = 484$ bits. For six rules, the total memory required will be $726 \times 4 = 2904$ bits. The proper choice in this design will use a 1024×4 static RAM to store knowledge. Matrix values (see Table 4) to be stored in memory are computed while in the FSIM and loaded into RAM from the host.

Control. The control unit represents the implementation's most complicated part. The control engine's main functions are to coordinate the proper operational sequence and to generate the memory address for accessing RAM during inferencing. The three count status blocks described earlier can be implemented using 4-bit synchronous counters.

The counter for rule-count resets at 0110_2 (the modulo-6 counter). U-count and V-count are used for indexing the two-dimensional implicational relation matrices, which reset at 1011_2 (the modulo-11 counter).

Each counter will output a ripple carry output (RCO) bit whenever the maximum count is reached. In addition to the three counters, a 10-bit counter (realized by cascading three 4-bit synchronous counters together) is needed to generate the RAM address. The RAM address generator resets when the RCO bits of all three status counters are set, which signifies that the maximum count (726) has been reached.

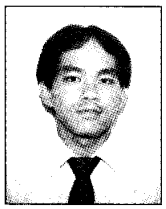
Incorporating reasoning systems on hardware is significant because future expert systems may have to make decisions in real time. This approach may prove indispensable in the near future, when sophisticated expert systems the size of pocket calculators become available.

We have presented a step-by-step approach towards realizing fuzzy expert systems in VLSI circuitry. Developing reasoning system hardware for an FPS consists of two stages. The initial stage specifies the algorithm in the form of fuzzy production rules. The algorithm is exercised and its functionality verified with the FSIM, a simulator for the FPS. Once the algorithm is deemed functional, the second stage designs special-purpose hardware to realize the system. Rules are converted to fuzzy relational matrices that can be programmed onto memory chips for use as knowledge during inferencing.

In addition, we have outlined a general block architecture for realizing the approximate reasoning processor. More importantly, we have shown that the reasoning system's knowledge base is easily reprogrammable and independent of the fuzzy implicational relation's interpretation used for inferencing. This is an extremely useful feature for accommodating reasoning systems in which the knowledge base is constantly changing, and enables a knowledge source to be made available as a plug-in module in future systems.

References

1. D. Dubois and H. Prade, *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, Orlando, Fla., 1980.
2. C.V. Negoita, *Expert Systems and Fuzzy Systems*, Benjamin/Cummings Publishing Co., Menlo Park, Calif., 1985.
3. L.A. Zadeh, "Making Computers Think Like People," *IEEE Spectrum*, Aug. 1984, pp. 26-32.
4. M. Togai and H. Watanabe, "Expert System on a Chip: An Engine for Real-Time Approximate Reasoning," *IEEE Expert*, Fall 1986, pp. 55-62.
5. E.H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," in *Fuzzy Reasoning and Its Applications*, E.H. Mamdani and B.R. Gaines, eds., Academic Press, Orlando, Fla., 1981, pp. 133-148.
6. M. Mizumoto and H. Zimmermann, "Comparison of Fuzzy Reasoning Methods," *Fuzzy Sets and Systems*, Vol. 18, 1982, pp. 253-283.
7. L.A. Zadeh, "A Theory of Approximate Reasoning," in *Machine Intelligence*, Vol. 9, J.E. Hayes, D. Michie, and L.I. Mikulich, eds., Elsevier, New York, N.Y., 1979, pp. 149-194.
8. M. Mizumoto, "Note on the Arithmetic Rule by Zadeh for Fuzzy Conditional Inference," *Cybernetics and Systems*, Vol. 12, 1981, pp. 247-306.
9. T. Whalen and B. Schott, "Issues in Fuzzy Production Systems," *Int'l J. Man-Machine Studies*, Vol. 19, 1983, pp. 57-71.
10. T. Whalen and B. Schott, "Alternative Logics for Approximate Reasoning in Expert Systems: A Comparative Study," *Int'l J. Man-Machine Studies*, Vol. 22, 1985, pp. 327-346.
11. D. Dubois and H. Prade, "Fuzzy Logic and the Generalized *Modus Ponens* Revisited," *Cybernetics and Systems*, Vol. 15, 1984, pp. 293-331.
12. L.A. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 3, 1973, pp. 28-44.
13. H.-J. Zimmermann and P. Zysno, "Latent Connectives in Human Decision Making," *Fuzzy Sets and Systems*, Vol. 4, 1980, pp. 37-51.
14. H.-J. Zimmermann and P. Zysno, "Decisions and Evaluations by Hierarchical Aggregation of Information," *Fuzzy Sets and Systems*, Vol. 10, 1983, pp. 243-260.
15. N. Rescher, *Many-Valued Logic*, McGraw-Hill, New York, N.Y., 1969.
16. M.H. Lim, *A Systematic Approach to Developing Silicon Inference Processors for Approximate Reasoning*, doctoral dissertation, University of South Carolina, Columbia. S.C., 1988.
17. K.J. Schmucker, *Fuzzy Sets, Natural Language Computations, and Risk Analysis*, Computer Science Press, Rockville, Md., 1984.
18. R. Degani and G. Bortolan, "The Problem of Linguistic Approximation in Clinical Decision Making," *Int'l J. Approximate Reasoning*, Vol. 2, 1988, pp. 143-162.
19. F. Eshragh and E.H. Mamdani, "A General Approach to Linguistic Approximation," in *Fuzzy Reasoning and Its Applications*, E.H. Mamdani and B.R. Gaines, eds., Academic Press, Orlando, Fla., 1981, pp. 169-187.
20. F. Wenstop, "Quantitative Analysis with Linguistic Values," *Fuzzy Sets and Systems*, Vol. 4, 1980, pp. 99-115.
21. G. Borriello, R.H. Katz, and A.G. Bell, "The Multibus Design Frame," Tech. Report No. UCB/CSD 85/232, University of California, Berkeley, Calif., May 1985.



Meng-Hiot Lim is a lecturer at Nanyang Technological Institute's School of Electrical Engineering (Singapore). He received his PhD in electrical and computer engineering from the University of South Carolina in 1988. His research interests focus on fuzzy logic, expert systems, VLSI CAD tools, and neural networks.

Yoshiyasu Takefuji is an assistant professor of electrical engineering at Case Western Reserve University. Before joining Case Western in 1988, he taught at the University of South Florida and the University of South Carolina. He received his BS (1978), MS (1980), and PhD (1983) in electrical engineering from Keio University (Japan). His current research interests focus on neural networks and parallel processing. A member of the IEEE Computer Society, ACM, and Neural Network Society, he received the Information Processing Society of Japan's best paper award in 1980. In 1989, he received the National Science Foundation's Research Initiation Award. More recently, he received a DARPA award (with Yoh-Han Pao, an *IEEE Expert* associate editor). He has published more than 40 technical papers and has written two books — *Digital Circuits*, Ohmsha, 1984; and *Neural Computing*, Baifu-kan, 1990 (both in Japanese).

The authors can be reached in care of Y. Takefuji, Glennan 515B, Case Western Reserve University, Cleveland, OH 44106.