VHDL and Fuzzy Logic If-Then Rules
Alex Zamfirescu and Cary Ussery
Vantage Analysis Systems, Inc.

June 18, 1992

Abstract

This paper explores the feasibility of using VHDL to model systems utilizing fuzzy logic. In particular, it deals with the representation and analysis of the behavior of collections of interacting objects, each of which is characterized by fuzzy if-then rules. The feasibility and desirability of using VHDL to model such systems is examined and an example is used to test out the ideas developed here. The paper concludes with a discussion of the importance and future trends in modeling fuzzy logic using HDLs, and VHDL in particular.

1 Introduction

Putting "fuzzy logic" and "VHDL" together might create a mental conflict for some people, especially those accustomed to clear cut, precise Boolean logic. Despite these psychological impediments, however, "fuzzy logic" and "VHDL" fit together quite well. The rich descriptive capabilities of VHDL [4] and the algorithmic power of fuzzy logic [2] complement each other. The reasons for this are:

- The robust, descriptive nature of VHDL maps well into the computational demands of fuzzy logic,
- VHDL supports the exact level of abstraction and information hiding required to make fuzzy logic implementations user-friendly,
- VHDL user-specifiable resolution functions and overloading capabilities map directly into a fuzzy logic paradigm, and
- modern fuzzy logic engineering requires support for hardware/software co-development, incremental modification of systems, reusability of modules, etc., all of which are enabled by VHDL.

Fuzzy logic has been gaining in acceptance and importance recently and is, finally, breaking out of its near-cult status into the mainstream of electronic system design. At the same time, VHDL has become an international standard for the design of digital systems. In spite of this, the literature lacks reports on the use of VHDL to model fuzzy systems, with the exception of [5]. This paper surveys some of the shortcuts VHDL can provide for fuzzy logic designers.

Fuzzy logic has been employed in a wide variety of consumer products and systems such as elevators, cranes, trains, ships and traffic controllers. In such applications what is actually employed is a small subset of fuzzy logic characterized as "Calculus of Fuzzy If-Then Rules," or CFR for short. Therefore, only the application of VHDL to CFR will be discussed in this paper; the appropriateness of VHDL for fuzzy logic in general will be presented elsewhere.

The paper begins with a description of fuzzy if-then rules including definitions and examples. This is followed by a section exploring the match between such rules (and systems based on these rules) and the VHDL language. Next, the prototype example of a reverse pendulum system is described. The paper concludes by identifying the main results of our work thus far and the directions and trends which might guide future work in this area.

2 The Fuzzy If-Then Rules

The behavior of some interacting objects is defined by fuzzy if-then rules when a collection of rules involving linguistic variables in terms of which object behavior is described is given. For example, "if pressure is low, then altitude is high" is such a rule. CFR provides the computational framework for analyzing such objects. The solution to the basic CFR problem has many ramifications reaching into control, decision analysis and expert systems domains.

2.1 Fuzzy Subsets, and Fuzzy Numbers

The concepts of fuzzy subsets and fuzzy numbers can be introduced in many ways. In fact there are more than 4000 papers and books on the theory and application of fuzzy numbers since Zadeh's first [2], and the field is growing exponentially. Only a rudimentary set of terminology pertinent to CFR will be presented here. For more details refer to [3]. Let E be a referential set (integer numbers Z or real numbers R). Ordinary subsets can be defined using the characteristic function

$$\forall x \in E : \mu_A(x) \in \{0, 1\}$$

showing belonging according to the value of the characteristic function, $\mu_A$.

1Stated as: given a system of objects each characterized by fuzzy if-then rules, what is the collection of fuzzy if-then rules that characterizes the system as a whole?
For the same referential E a fuzzy subset A will be defined by a characteristic function taking values in the interval \([0, 1]\).

\[ \forall x \in E : \mu_A(x) \in [0, 1] \]

A fuzzy subset of \(R\) is normal if the highest value of \(\mu_A(x)\) is 1.

A fuzzy subset \(A \subseteq R\) is convex if and only if every ordinary subset

\[ A_\alpha = \{ x | \mu_A(x) \geq \alpha \}, \alpha \in [0, 1] \]

is convex (i.e., does not contain disjoint intervals). A fuzzy number in \(R\) is by definition a fuzzy subset of \(R\) that is convex and normal.

It is a consequence of the previous definitions that for any fuzzy number, if the level of presumption (see Figure 1) increases the internal of confidence never increases.

Exact numbers, or crisp values, can be considered as particular fuzzy numbers whose characteristic function takes value 1 for only one element and value 0 elsewhere.

In CFR descriptions each fuzzy number is associated to one or more linguistic variables from a set \(L = \{ L_1, \ldots, L_n \}\). However, for the purpose of this initial discussion the linguistic variables and the associated fuzzy numbers they denote will be used interchangeably. Later, when VHDL will be used for implementation, the distinction will become evident.

Equality for fuzzy numbers is in general defined as

\[ [N = M] \stackrel{def}{=} \sup_{\alpha} (\min(\mu_N(x), \mu_M(x))) \]  

This equality definition is valid for any combination of crisp and/or fuzzy numbers. For the case of two crisp numbers it duplicates the normal equality. The combination of crisp and fuzzy numbers appears for example when real values are monitored by fuzzy controllers.

\[ \text{Figure 1 - A Fuzzy Number} \]

2.2 Fuzzy And and Fuzzy Or

Typical CFR rules involve fuzzy and and fuzzy or operations. These operations can be defined using associative real norms that may be extended to any number of arguments. Such norms are maps \([0, 1]^n \rightarrow [0, 1]\) associating pairs of subunitary reals to subunitary reals. Typically a pair of norms are needed with one representing the "fuzzy and" and the other representing the "fuzzy or." A classic example uses "min" for "fuzzy and" and "max" for "fuzzy or."

2.3 A Fuzzy If-Then Rule

Suppose we have the following if-then rule as part of a controller

\[ \text{if } (\theta = \text{EQUILIBRIUM}) \quad \text{and} \]
\[ \quad (d\theta = \text{LARGE_NEGATIVE_D_THETA}) \quad \text{then} \]
\[ \quad \text{drive\_motor} = \text{LARGE_POSITIVE_CURRENT} \]

where EQUILIBRIUM, LARGE_NEGATIVE_D_THETA, and LARGE_POSITIVE_CURRENT are fuzzy numbers, \(\theta\) and \(d\theta\) are crisp numbers, and \(\text{drive\_motor}\) represents the controlled output.

How are such rules evaluated in CFR? First, the equality \((\theta = \text{EQUILIBRIUM})\) is evaluated returning a real number \(a_1\). Second, the equality \((d\theta = \text{LARGE_NEGATIVE_D_THETA})\) is evaluated and \(a_2\) is returned. Third, the "fuzzy and" norm computes the minimum \(\alpha\) of \(a_1\) and \(a_2\). This \(\alpha\) value is the value of the antecedent for the rule.

If there are more rules that have as a consequence LARGE_POSITIVE_CURRENT for \(\text{drive\_motor}\) the values of their antecedents are combined using the "fuzzy or" norm (typically "max"). The result value is associated with LARGE_POSITIVE_CURRENT in a "voting" set for the value of \(\text{drive\_motor}\). This set might look like

\[ \{ \text{LARGE_POSITIVE_CURRENT}, \text{ZERO_CURRENT}, \ldots \} \]

Note that the set contains pairs of crisp and fuzzy numbers.

2.4 Defuzzifiers

The general form of the "voting" set is

\[ \{ \beta_1, \beta_2, \ldots, \beta_m \} \]

where \(\beta_i\) are crisp and \(M_i\) are fuzzy.

This set can be defuzzified into a crisp or a fuzzy number. A commonly used crisp defuzzifier called center of mass defuzzifier will return the value

\[ \delta = \frac{\sum \beta_i c_{vi}}{\sum \beta_i} \]

where \(c_{vi}\) is the central value (where \(\mu_{M_i}(x) = 1\)) of \(M_i\).

If fuzzy arithmetic\(^2\) is performed in the above formula and \(c_{vi}\) is replaced by the fuzzy numbers \(M_i\) then a defuzzifier returns a fuzzy result.

\(^2\)Addition, subtraction, multiplication, and division can be easily defined for the fuzzy numbers.
2.5 Fuzzy If-Then Rules Execution Computational Requirements

The execution or simulation of the behavior described by a collection of fuzzy if-then rules requires the following:

- real number manipulation,
- ability to represent fuzzy numbers and manipulate them via linguistic variables,
- association ability for crisp and fuzzy numbers (the "voting" set),
- ability to hide computational details (like computing the result of an equality or an "and" operation),
- ability to resolve a set of elements to one crisp or one fuzzy value (defuzzifiers),
- ability to evaluate rules based on changes and to predict values as consequences,
- ability to change fuzzy number belonging either manually or automatically as a result of a neuronal computation.

The following sections present how VHDL matches each of these requirements.

3 VHDL Matches Fuzzy

3.1 Real Number Manipulation

One of the most controversial feature of VHDL, its floating point types, happens to be a requirement for fuzzy logic. The portability issue related to the host precision is meaningless for fuzzy logic. It is true that the support (the domains of the characteristic function) is discrete for some fuzzy numbers, and some fuzzy implementations use integer representations of the degrees of belonging, but without its Std.STANDARD.REAL type, VHDL would probably not qualify as a fuzzy logic implementation candidate.

3.2 Fuzzy Number Representation

In representing fuzzy numbers in VHDL the designer have more choices:

1. To encode the belonging function in tables (constant or variable arrays) for an entire integer or discrete domain;
2. To use triangular fuzzy numbers, storing only the breaking points in arrays. Standard functions accepting enumeration literals corresponding to each fuzzy linguistic variable are used to compute belonging (in fact equality) in this case;
3. To define VHDL functions that compute characteristic functions and use enumeration types or constants for the corresponding fuzzy linguistic variables;
4. To use value systems (enumeration types) and pre-computed two dimensional tables of real numbers to represent the fuzzy operators. Tables can be stored in constant arrays, or if belonging migration is needed in variable arrays or signal arrays declared in packages.

Choice 2 above was used in the example discussed latter in Section 4. Note that the last choice above (number 4) is similar to some value systems actually used in (digital) circuit design. The broad spectrum of declarable composite VHDL types, including arrays indexed by enumeration types makes encoding of fuzzy numbers an easy task. Constants with meaningful names or enumeration literals can be used as fuzzy linguistic variable. Simple mechanisms based on arrays or case statements can bring the characteristic function information when needed.

3.3 Associating Crisp and Fuzzy Numbers

VHDL records can hold crisp numbers in REAL types fields and fuzzy numbers in appropriate fields of an enumeration type, discrete type, or array type. Unconstrained arrays of records can be used to model the "voting" set containing pairs of crisp and fuzzy numbers.

3.4 Fuzzy Operators

The computational details of the fuzzy equality are meaningless for the fuzzy logic expert entering the application specific rules. In VHDL it is possible to hide these details by providing an easy to understand abstract notation for the user entering fuzzy rules.

The implementation of the fuzzy number equality can be done for example

- by overloading the equality operator "=",
- by defining a named subprogram,
- or it can be a simple fast table look-up.

The use of a fuzzy VHDL package containing subprograms, overloading the "and", "or" and "not" predefined operators by fuzzy operators, can make the human interface to the fuzzy system as friendly as that provided by any special purpose fuzzy tool.

3.5 Defuzzification

Resolution functions are ideal mechanisms provided by VHDL for resolving a set of "voting" elements to one crisp or one fuzzy value. An unconstrained array of records as parameter to the resolution function is used for holding the set of "voting" pairs. Modeling the set as an array is transparent since defuzzifiers are reduce operators that are source order invariant [6].

The resolution function can incorporate the "or" operator together with the defuzzification or it could contain only the later. Other choices for defuzzification in VHDL use components or processes, both introducing a delay of at least one cycle.
3.6 Fuzzy Rule Specification

Fuzzy rules semantics require continuous evaluation, but the evaluation based on changes gives the same behavior. Hence the VHDL signal assignments are the best candidates for rule specification. The future value predicted as a consequence of a rule evaluation can be encoded as in

\[
\text{drive\_motor} \leftarrow (\text{LARGE\_POSITIVE\_CURRENT}, \\
(\theta = \text{EQUILIBRIUM}) \text{ and } \\
(d\_\theta = \text{LARGE\_NEGATIVE\_D\_THETA}) )
\]

3.7 Belonging Migration

VHDL gives the ability to change fuzzy number belonging. This can be done either manually or automatically. For the manual change of the belonging generics or deferred constants are recommended. By using variables to hold belonging information and a converging computation to modify these variables in a loop including the fuzzy rules evaluation and some optimum criteria, one can easily model a fuzzy system controlled by a neuron net.

4 Example

In order to verify some of the ideas presented here, we chose to implement an example of a fuzzy controller system using VHDL. A number of goals were identified for the example. These were listed in [5]. The overall goal was to verify the ease of modeling fuzzy systems in VHDL.

Figure 2 - The Reverse Pendulum System

4.1 The Reverse Pendulum Example

We selected the "reverse pendulum" problem as a straightforward representation of our ideas. In this example, a bob on a shaft must be kept balanced on top of a motorized platform. When the bob is disrupted the motor is used to move the platform back and forth (our example uses only two dimensional space) to keep the bob balanced. Figure 2 shows the architecture of the system. On the left is the fuzzy controller module and on the right is the pendulum subsystem which contains the physical model of the pendulum movement and some display interface.

4.2 A Fuzzy Logic Package

One of the goals in developing a prototype system was to begin developing a set of general routines and data structures which could be reused in the development of subsequent systems. The VHDL package is an ideal mechanism for modularizing data type definitions and subprograms. Our package defines two types of functions. First, there is a general-purpose set of "belonging" functions which are used to implement "how much" a given real number belongs to "belonging" set (see above). For instance, the following function is defined:

\[
\text{function bell (} \\
\text{zeroBreakPoint1: REAL; } \\
\text{oneBreakPoint : REAL; } \\
\text{zeroBreakPoint2 : REAL)} \text{ return REAL;}
\]

The second class of functions are overloaded definitions of AND, OR, and NOT which, in real-valued fuzzy logic, are implemented by the functions MIN, MAX, and 1-X. For example,

\[
\text{function "and" (L,R:F-Real) return F-Real is} \\
\text{begin} \\
\text{if } L > R \text{ then return } R; \\
\text{else return } L; \\
\text{end if;}
\]

4.3 Abstract Types

In order to represent fuzzy data accurately and pneumonically, abstract types are used. For instance, our defuzzification depends on the location of a real value over a spectrum and the weight at which this value is contributing to the overall value. The following record definition is used to aggregate these values into individual objects:

\[
\text{type FUZZY\_CURRENT is (} \\
\text{LARGE\_NEGATIVE\_CURRENT, } \\
\text{SMALL\_NEGATIVE\_CURRENT, } \\
\text{ZERO\_CURRENT, } \\
\text{SMALL\_POSITIVE\_CURRENT, } \\
\text{LARGE\_POSITIVE\_CURRENT);}
\]

\[
\text{type CONTROL is record} \\
\text{with\_fuzzy: FUZZY\_CURRENT; } \\
\text{whenever: REAL;}
\]

4.4 Behavioral Calculations

The physical model of the pendulum movement contains non-trivial calculations of real numbers. The calculation consists of determining consecutive angles of the shaft based on the torque and inertia of the ball movement. VHDL allows for complex expressions and statements. By using complex expressions, the new angle is calculated in only four lines:
Torque := (MassOfBall/100.0) * g * 
SizeOfShaft * cos(NewAngle) + 
SizeOfMotor * 4.0 * drive_motor;

Inertia := SizeOfShaft * SizeOfShaft * 
MassOfBall / 1000.0;

ChangeInAngle := ChangeInAngle + 
(Torque/Inertia) * dT;

NewAngle := NewAngle + 
(ChangeInAngle + dT) + 
(Torque/Inertia) * dT_squared;

4.5 Configurations

Another goal which was maintained throughout the development of the example was to make the system easy to modify and allow for the easy insertion and test of design alternatives. Configurations were used to put together the system. At any point, the configuration could be changed to try alternatives or attach debugging code. In addition, the controller was made "generic" about the number of rules which could be added. To add (and remove) rules we used generic parameters, generate statements, and indexed configurations. The latter allowed us to bind a rule template to a specific rule to be used in the system (in our case we had eleven rules contributing to the movement of the pendulum.

for Controller: Fuzzy-Controller use 
entity WORK.Fuzzy-Controller(RULE-BASED) 
generic map (numberOfRules => 11) 
port map (theta, d-theta, output);

for RULES(1) use 
entity WORK.Fuzzy-Entity(RULE-1) 
port map (theta, d-theta, drive_motor); 
end for;

... -- Bind All Rules 
end for;

4.6 Graphics Interface

The initial development of the pendulum system relied on the use of TEXTIO to interact with the system and view results. However, it was hard to visualize the movement of the pendulum by reading large sets of real numbers. A graphics system was developed using X Windows interacting with the simulation model through the Vantage's STYX C interface. We switched in a new architecture for the pendulum display using the system configuration declaration. With a graphics environment it was much easier to see how the system was working and allowed us to develop a more intuitive interface to the system in general. A depiction of the graphical display of the system is shown in Figure 3.

4.7 Experiences

We found that the concepts of CFR mapped well into VHDL. The system we developed was up and running within a matter of days and we were able to try out and debug a number of rules. Our familiarity with VHDL enabled us to make use of its rich feature set. The most surprising result was that we had no trouble finding a mapping between some concept in fuzzy logic into VHDL in most cases we found multiple mappings. Using similar modeling techniques, we feel designers and researchers of fuzzy logic systems would benefit from using VHDL to explore design alternatives and develop rule sets.

5 Conclusions

Assuming a familiarity with both VHDL and fuzzy logic, the mix of the two seems quite "natural". This appears to be due to the close match between the computational and functional requirements of fuzzy systems and the expressive power of VHDL.

This work could be expanded in the future to cover more of the concepts involved in developing fuzzy systems. In addition, work in the area of hardware/software co-development, synthesizable fuzzy descriptions, and adaptive fuzzy controllers should be pursued. Our experience here has given us confidence that further work in the area will benefit fuzzy logic engineering and extend the application domain of VHDL.

6 Acknowledgements

The authors would like to express their gratitude to Dr. Jean Mermet for helpful discussions on this topic. We are indebted to Rick Lazansky and the rest of our group at Vantage for their enthusiastic support. We also thank Graham Shenton for his valuable comments, and the anonymous reviewers for their helpful suggestions for improving the manuscript.

References


