VHDL framework for modeling fuzzy automata

Doru Todinca, Daniel Butoianu
Department of Computers
Politehnica University of Timisoara, Romania
Email: doru.todinca@cs.upt.ro

***Abstract*—Fuzzy set theory has been used to extend different domains of mathematics and also domains from applied sciences and from engineering, such that now there exists fuzzy logic, fuzzy arithmetic, fuzzy expert systems, fuzzy logic controllers (FLCs), fuzzy automata, fuzzy flip-flops, etc.**

In this work we concentrate on fuzzy automata (FA), which are fuzzy logic extensions of crisp automata (or finite state machines). Many researchers have noticed that, while crisp automata are widely used, being incorporated in almost any technical device, fuzzy automata have very few practical applications, being rather a theoretic concept. Among the factors that contributed to this situation we can enumerate: 1) the lack of controllability of FA (in certain conditions the degree of membership of the states of FA decrease towards zero and cannot be increased after that) and 2) there are too many types of fuzzy automata.

We developed a VHDL framework that can be used for modeling fuzzy automata and for investigating their performance.

In this paper we use several examples of FA from literature in order to illustrate how our VHDL framework can investigate the efficiency of different functions used for computing the degree of membership of the next state of a FA. We also show that some techniques proposed in literature for improving the performance of FA (more precisely, to avoid the degree of membership of FA states to decrease toward zero) are not effective on the fuzzy automata from these examples.

I. Introduction

Fuzzy logic has gained acceptance in many domains of science and engineering, being used now in control engineering, telecommunications, computer engineering, pattern recognition, information retrieval, linear and non-linear programming, etc. There are many industrial products that incorporate fuzzy logic, from everyday electronic products like washing machines, handset cameras, vacuum cleaners, to unmanned subway trains and chemical plants.

Most of these products are based on fuzzy inference, which means combining facts with rules expressed in (a subset of) a natural language in order to obtain control actions. When the time constraints of an applications are hard, fuzzy inference is implemented in hardware, such a circuit being named fuzzy logic controller (FLC).

No matter how complex its implementation is, a FLC's outputs depend only on its current inputs, no previous history is taken into account and no FLC 'state' is considered. It means that a FLC behaves like a combinational (i.e. a memoryless) circuit. Many applications demand the existence of states, the behaviour of a device being determined not only by its current inputs, but also by its states.

From the fuzzy logic point of view, we can imagine that the states of such a device can be expressed by fuzzy sets, which bring us to the idea of fuzzy automata (FA), that are fuzzy extensions of classical automata (finite state machines).

Fuzzy automata are a hot research topics, as shown by the recent publications that deal with theoretical aspects: construction of FA from regular expressions in [1], bisimulations for FA [2], nondeterministic fuzzy automata in [3].

There are also applications of FA recently reported: Schmidt and Boutalis ([4]) use fuzzy discrete event systems (FDES) in combination with multi-objective control for robot navigation; Wu, Pang and Han ([5]) apply FA for target recognition based on image processing, and Bailador and Triviño use temporal fuzzy automata for pattern recognition ([6]).

An FPGA implementation of the generalized fuzzy automata (GFA) presented in [7] is reported in [8].

In [9] Chen proposes a new class of FA, that he calls Generalized Fuzzy Automata, or GFA. The GFAs belong to the class of FA with fuzzy inputs and states, where the time is also fuzzy and continuous. With GFAs, he develops a method of feedback fuzzy control 'with words' in [10]. Then, Chen et al apply their method of fuzzy control with GFA to telecommunications problems, more precisely to Quality of Service (QoS) improvement in Wireless LAN networks: [11] and [12].

However, as observed by Virant and Zimic in [13], classical finite state machines (i.e. automata) are much more extensively used in practical applications than fuzzy automata. Virant and Zimic expect that the role of fuzzy automata will increase.

We believe that there are a number of factors that prevent fuzzy automata to be widely spread in engineering applications:

- Non-controlling behaviour: in certain conditions the degree of membership of a fuzzy state can decrease continuously towards zero, which is obviously not a desired behaviour. The problem has been reported for example in [14], [15]. Different techniques have been proposed in order to solve this problem: peak hold in [14], conservation of state value in [15], or state normalization in [16].
- There are too many types of fuzzy automata: since fuzzy automata are fuzzy extensions of classical automata, different researchers have proposed different ways to extend classical (crisp) automata through the framework of fuzzy logic. A possible classification of fuzzy automata can be found in [9], but there are also other possibilities to classify fuzzy automata. This problem, of different extensions of a classical concept, is very common in the fuzzy set theory: there are classes of operators for set operations

like union, intersections and complement, different types (formulae) for fuzzy inference, for defuzzification, etc. While in fuzzy inference and fuzzy domains with a longer history of applications, some of these different concepts have imposed over other similar concepts, in the domain of FA the practitioneers are still confused by the pletora of possible fuzzy automata.
- Maybe another confusing issue related to FA is the relation between automata and flip-flops: while classical flip-flops are building blocks in the design or synthesis of automata, there is no such relation between fuzzy automata and fuzzy flip flops. Some investigations in the attempt to link fuzzy automata and fuzzy flip-flops have been reported in [17]. In this work we will concentrate on FA, leaving the relation between FA and fuzzy flip-flops for further investigations.

Here we propose a framework that will help the investigation of fuzzy automata. Our framework will allow the study of the behaviour of different fuzzy automata in different circumstances and to use different techniques for solving the problem of state decreasing membership function.

The rest of the paper is organized as follows: next section contains a brief description of fuzzy sets and fuzzy automata, section III described the VHDL implementation of our framework, section IV shows some simulation results, and the paper ends with a section of conclusions and future developments.

II. Fuzzy sets. Fuzzy automata

A. Fuzzy sets

Fuzzy set theory was introduced by L.A. Zadeh in 1965. Fuzzy sets extend the notion of classic, or crisp sets in the sense that, while for a crisp set, an element either belongs or it does not belong to that set, for a fuzzy set, an element can belong to that set *in a certain degree*.

More formally, given an universe of discourse X (a crisp set) and a crisp subset $A \subset X$, for each element $x \in X$, either $x \in A$ or $x \notin A$.

According to [18], a fuzzy set $\tilde{A} \subset X$ is the set of ordered pairs

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) | x \in X\}$$

where $\mu_{\tilde{A}}(x) : X \to [0,1]$ is called membership function or degree of membership. When the closed real interval [0, 1] is replaced with the discrete set $\{0,1\}$, then the fuzzy set $\tilde{A}$ becomes a crisp set.

With fuzzy sets in the same universe of discourse we can perform operations of intersection, reunion and complement. Zadeh defined the union of two fuzzy sets as the maximum between their membership functions, the intersection, as the minimum of their membership functions, and the complement of a fuzzy set $\tilde{A}$, the set having the membership function $1 - \mu_{\tilde{A}}(x)$.

The minimum and maximum operations have been extended by t-norms and s-norms. From [18], page 30, t-norms are two-valued functions defined on $[0,1] \times [0,1] \to [0,1]$ with the properties:

1) $t(0,0) = 0$, $t(\mu_{\tilde{A}}(x), 1) = t(1, \mu_{\tilde{A}}(x)) = \mu_{\tilde{A}}(x), x \in X$
2) monotonicity
3) commutativity
4) associativity

Similarly (see [18]) can be defined s-norms, or t-conorms, as two valued functions defined on $[0,1] \times [0,1] \to [0,1]$ with the properties:

1) $s(1,1) = 1$, $s(\mu_{\tilde{A}}(x), 0) = s(0, \mu_{\tilde{A}}(x)) = \mu_{\tilde{A}}(x), x \in X$
2) monotonicity
3) commutativity
4) associativity.

Examples of t- and s-norms:

- drastic product t_w and drastic sum s_w:

$$t_w(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \begin{cases} \mu_{\tilde{A}}(x), & if\ \mu_{\tilde{B}}(x) = 1 \\ \mu_{\tilde{B}}(x), & if\ \mu_{\tilde{A}}(x) = 1 \\ 0, & if\ \mu_{\tilde{A}}(x) < 1 \\ & and\ \mu_{\tilde{B}}(x) < 1 \end{cases}$$

$$s_w(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \begin{cases} \mu_{\tilde{A}}(x), & if\ \mu_{\tilde{B}}(x) = 0 \\ \mu_{\tilde{B}}(x), & if\ \mu_{\tilde{A}}(x) = 0 \\ 1, & if\ 0 < \mu_{\tilde{A}}(x) \\ & and\ 0 < \mu_{\tilde{B}}(x) \end{cases}$$

- bounded difference t_1 and bounded sum s_1:

$$t_1(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = max\{0, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - 1\}$$

$$s_1(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = min\{1, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)\}$$

- algebraic product t_2 and algebraic sum s_2:

$$t_2(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

$$s_2(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)) = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

- min and max (minimum and maximum).

B. Fuzzy automata

According to [7], automata in general are computational systems having inputs, states, outputs, a function that computes the transition from the current state to next state (transition function) and a function that computes the outputs based either on the current state (Moore automata), or based on the current state and the input (Mealy automata). Automata may have also initial and final states. The classic automata can be deterministic, non-deterministic and probabilistic.

Fuzzy logic has been used to extend the classical automata, resulting a large variety of fuzzy automata (FA). The states of FA can be continuous (e.g. fuzzy automata with fuzzy relief [13]), or discrete: fuzzy sequential circuits ([15], [16]), a medical state monitor ([14]), fuzzy state machine ([19], based on [20]), generalized fuzzy automata [7]. Even the time can be fuzzy, as emphasized in [21]. Chen (in [9]) proposed a classification of FA depending on the fuzzy or crisp character

of their inputs, states, and time, and also based on the discrete or continuous nature of the time.

In this work we will study only the discrete synchronous fuzzy automata, but we plan to extend our framework in order to include for example fuzzy automata with fuzzy relief ([13]), expressed in a discretized form.

Our model of fuzzy automata is based on the descriptions from [15], [19] and [7], trying to 'unify' these approaches. Also, in this work we do not investigate the computation of the outputs of the FA and focus only on the evolution of the states of FA. However, our framework contains the implementation of the part that determines the outputs of the FA.

In [15] a fuzzy automaton is defined starting from the sets of its inputs U, states X and outputs Y, that are called universes of discourse. The inputs, states and outputs of the FA are fuzzy sets in the universes U, X and Y.

For every input u^j a transition matrix $M(u^j)$ is defined:

$$\mathbf{M(u^j)} = \begin{pmatrix} \mu_x(x_{t+1}^1|x_t^1,u^j) & \dots & \mu_x(x_{t+1}^m|x_t^1,u^j) \\ \vdots & \ddots & \vdots \\ \mu_x(x_{t+1}^1|x_t^m,u^j) & \dots & \mu_x(x_{t+1}^m|x_t^m,u^j) \end{pmatrix}$$

Here $\mu_x(x_{t+1}^i|x_t^k,u^j)$ is the degree of transition from state x_t^k to state x_{t+1}^i when the input u^j is applied (at the moment t) and all μ_x are fuzzy values (in the interval $[0,1]$). The authors of [15] introduce the idea of conservation of state, which means to take into account in the computation of the next state not only the value (degree of membership) of each input, but also the negated value of each input. Using the notations from ([15]), the formula for next state will be:

$$\mu_x(x_{t+1}) = \sum_{x^i}\sum_{u^j}[\mu_x(x_t)\times\mu_u(u_t)\times\mu_x(x_{t+1}|x_t,u_t)$$

$$+\mu_x(x_t)\times\overline{\mu_u(u_t)}\times\mu_x(x_{t+1}|x_t,\overline{u_t})]$$

The symbols $\sum$ and $+$ denote here algebraic sum, the symbol $\times$ represents algebraic product, and $\overline{\mu_u(u_t)}$ represents the membership degree of the complement (negation) of input u (at step t). The sums are for all states x^i and all inputs u^j. All matrices $M(\overline{u^j})$ are considered in [15] to be the unit matrix of the corresponding dimension and are called inactive paths.

It is common in literature to use for μ_x combinations of maximum and minimum, but Mori, Otsuka and Mukaidono ([15]) propose to use algebraic sum and algebraic product instead of maximum and minimum. Here we go one step further and, based on an idea from [7], use two general functions, $F1$ instead of minimum (or algebraic product), and $F2$ instead of maximum (or algebraic sum). In [7] $F1$ is called the membership assignment function, and $F2$ is called the multi-membership resolution function. In our case, $F1$ is a parametrized function that can be any t-norm, while $F2$ is a parametrized function that can be any s-norm.

III. VHDL IMPLEMENTATION

In our framework we want to test different approaches to fuzzy automata, to use different formulae for fuzzy composition, t-norms and s-norms, hence the flexibility of the implementation is a primary goal.

We have chosen VHDL (the achronym stands for Very High Speed Integrated Circuits Hardware Description Language) as a modeling language for the following reasons:

1) being a hardware description language, VHDL can be used for both simulation and automatic synthesis, allowing us to obtain a hardware implementation (e.g. on a FPGA) of a fuzzy automaton. In this way we will be able to evaluate the hardware properties of the circuit: input-output delay, clock rate, occupied surface, etc.
2) VHDL is a high level programming language, which permits a behavioral description of a system at a high level of abstraction. Also, it is possible to simulate the behaviour of both the controller (FA) and the controlled application.
3) using VHDL, the behaviour of a FA can be represented in different ways, for example as a signal diagram, or to be dumped in a file in order to support further processing.

In order to obtain a high degree of flexibility, we use a structure of VHDL packages. A *package* contains a collection of declarations that can be made visible to other design units.

The first package, named `Constants` defines the constants needed in the framework. We have denoted with *X* the number of states of the fuzzy automata, with *U*, the number of inputs , and *Y* is the number of outputs.

In the package `Types` we define the VHDL types used in the framework. The types are based on the constants defined in the previous package. First a fuzzy value type (degree of membership) *fv* is defined as a subtype of real numbers, having values in the interval $[0,1]$. Starting from *fv*, we define the types X_type, U_type Y_type as arrays of *fv* with dimension *X*, *U*, *Y*, for the states, the inputs and the outputs of the fuzzy automata. Also, multidimensional arrays of fuzzy values are defined. For transition matrix we define a type UXX_type, which is an $U\times X\times X$ array of *fv*. Similar types are used for computing the next state and the output of the fuzzy automata.

A third package, `Functions`, which is based on the previous two packages (i.e. `Constants` and `Types`) contains the functions used. For t-norms and s-norms we use a parametrized implementation (i.e. the functions $F1$ and $F2$), which allows the selection of a t-norm (or s-norm) from a set of such functions. The t-norms and s-norms shown in subsection II-A are implemented here. This package contains also other functions, necessary to compute the next state and the outputs of the automaton. The functions necessary for implementing the conservation of state and the state normalization techniques are described here as well.

The specific data for a certain fuzzy automaton is read from a text file by a procedure named `initialize_FA()`. It returns the initial state of the FA and its transition matrix.

The initial state will be stored in a variable $Init\_States$ of type X_Type, while the transition matrix will be stored during the simulation in a variable called $Paths$ of type UXX_type. For modeling a FA with conservation of states we use the variable $Inactive\_Paths$, also of type UXX_type. The variable $Inactive\_Paths$ is generated by a function named `create_Inactive_Paths`.

In order to simulate in VHDL a circuit or a system, we describe it in terms of design units. The most important design units are the *entity* and the *architecture*. The entity specifies the name of the circuit and its interface (ports and parameters, called generic parameters), while the actual functionality of the circuit is described in its architecture. An entity can have any number of architectures, feature that allows different views (e.g. behavioral or structural) of that entity. When an entity becomes a component in a bigger architecture, its generic parameters can be changed in the component instantiation statement.

The VHDL code that describes the FA circuit entity is here:

```
Entity FM_circuit is
Generic (
with_conserving_transition: natural:=1;
out_file: string:="out.txt";
tf1, tf2: integer;
fa_type_state: fa_type);
--the ports of the circuit
  Port(reset_i, clk_i: In bit;
with_state_normalisation: in bit;
--...);
End Entity;
```

The name of the circuit is $FM\_circuit$ and its generic parameters are:

1) $with\_conserving\_transition$: it can be 1, when the conservation of the state is applied, or 0, when it is not applied.
2) out_file: it is the text file where we dump the results of the simulation. The name of the file can be changed.
3) $tf1$, $tf2$: determines which t-norm to use for the function $F1$ and which co-norm to use for the function $F2$.
4) fa_type_state: determines which kind of FA to use. For the moment we have implemented only the 'Crafter' ([19]) and the 'transition matrix' ([15]) kind of FA and the purpose was to check by simulation that these two types of fuzzy automata are identical, i.e., they are rather different description styles of the same FA. In the future we plan to implement other types of FA, e.g. fuzzy automata with fuzzy relief ([13]).

In the current implementation the feature of state normalization is modeled as a port of the circuit, not as a generic parameter. The idea is that in this way we can apply state normalization only at certain moments, e.g., not necessarily after each change of state. Had the parameter been a generic, the state normalization function would have been either active, or inactive for the entire duration of the simulation. Other input ports of the circuit are the reset ($reset\_i$) and clock ($clock\_i$) signals.

When the reset is active the circuit (FA) goes to the initial state. It calls the procedure `initialize_FA` and the function `create_Inactive_Paths`.

We use the FA from [19] in order to illustrate the functioning of our VHDL code. The VHDL code that contains the values for the variables `Init_States` and `Paths` is given below:

```
Paths:= (( (0.0, 0.4, 0.2, 1.0),
(0.3, 1.0, 0.0, 0.2),
(0.5, 0.0, 0.0, 1.0),   --M(u1)
(0.0, 0.0, 0.0, 1.0)),

  ( (0.0, 0.0, 1.0, 0.0),
(0.2, 0.0, 0.0, 1.0),
(0.0, 0.0, 0.0, 1.0),    --M(u2)
(1.0, 0.3, 0.0, 0.6)));

Init_States:=(1.0, 0.8, 0.6, 0.4);
```

In this case the initial state is $x1$ with a degree of membership 1.0 (full membership), $x2$ with a degree of 0.8, $x3$ with 0.6 and $x4$ with a degree of 0.4.

When a fuzzy input vector $U_1 = (u1\ u2)$ is applied at the moment t_1 to the input of FA, a fuzzy composition between the vector U_1 and the variable $Paths$ is performed. It results an $X \times X$ (4×4 in this case) state transition matrix that describes the transition from the current state of the automaton to the next state. For example, if the input vector has the membership values $U = (1.0\ 0.4)$ and max-min composition is applied, then the state transition matrix T_{U_1} will be:

$$\begin{array}{c} \\ x_{P_1} \\ x_{P_2} \\ x_{P_3} \\ x_{P_4} \end{array} \begin{array}{c} \begin{array}{cccc} x_{N_1} & x_{N_2} & x_{N_3} & x_{N_4} \end{array} \\ \begin{pmatrix} 0.0 & 0.4 & 0.4 & 1.0 \\ 0.3 & 1.0 & 0.0 & 0.4 \\ 0.5 & 0.0 & 0.0 & 1.0 \\ 0.4 & 0.3 & 0.0 & 1.0 \end{pmatrix} \end{array}$$

In order to determine the next state of the FA, a composition between the current state vector ($Init\_States$ in this case) and the matrix T_{U_1} is performed. With max-min composition it results the next state vector: X_N (or X_{t+1}) $= (0.5\ 0.8\ 0.4\ 1.0)$. For more details about the computations please refer to [19], or to [20], chapter 12, section 12.6 "Fuzzy automata".

The state is changed only when the $clock\_i$ changes its value from '0' to '1', i.e, at the rising edge of the clock signal.

The circuit is tested in a test bench where the clock and reset signals are provided by a clock and reset generator. Also, the inputs of the FA are given either manually (e.g. for debugging), or are read from a text file, in order to automatize the simulation. In order to obtain in parallel the results of several simulations, we use the VHDL instruction *Generate* that instantiates several times the $FM\_circuit$ with different values of its generic parameters. Also, the name of the output file is different for each instantiation. For example,

we collected in parallel the evolution of the states of the FA with different t-norms and s-norms, and with and without conservation of the state. Some results are shown in the next section.

IV. Results

A. General aspects

In this section we will illustrate what kind of investigations can be performed using our framework. First we verified the correctness of the implementation on the numerical examples presented in [15] and [19]. We have also verified that the 'kind' of automata 'Crafter' ([19]) and 'transition matrix' ([15]) are identical, being only different description styles of the same FA. We consider that these were only preliminary simulations and do not present detailed results for them.

We aimed to study the following problems:

1) the efficiency of different operations (t-norms and s-norms) used for FA. For the functions $F1$ we have used different t-norms, while for function $F2$ we have used fuzzy s-norms. We consider that the operations are efficient if the fuzzy automaton is controllable,i.e., the degree of membership of its states can be influenced by the FA's inputs. The FA is not controllable if the degree of membership of its states remain blocked at certain values (e.g. zero) and cannot be changed from the inputs. This problem is often reported for fuzzy automata and fuzzy flip flops.
2) the effectiveness of some of the methods proposed in literature in order to overcome the lack of controllability of FAs. From the existing methods we have implemented by now the conservation of state ([15]) and the normalization of state ([16]).

For the first problem we used for the functions $F1$ and $F2$ different pairs of s-norms and t-norms:

1) maximum and minimum, which are the most used in literature for FA.
2) drastic sum and drastic product
3) algebraic sum and algebraic product
4) bounded sum and bounded product

We implemented four examples of fuzzy automata from literature and tested them using our framework:

1) the FA described in [15]
2) the 'Crafter' FA, from [19] and [20].
3) the FA described in example 1 from [7]
4) the FA described in example 4 (Figure 5) from [7].

The first FA (from [15]) is too small, having only two states and one input. We used it mostly for checking the correctness of our VHDL code and we do not present detailed simulation results for it.

The fourths FA (example 4 from [7]) has a transition matrix with very few non-zero values, and hence, for the majority of the simulations that we performed with it, the degree of membership of all its states quickly go to zero and cannot be changed after that. This example is not considered relevant in this stage of our work, but it might require more attention from us in the future. We do not present detailed simulation for it, either.

The second FA ('Crafter', from [19]) and third FA, (i.e. example 1 from [7]), behave very similarly in the sense that, for the functions for which the one of them was controllable, so was the other FA, while for the cases when one of them remained stuck in certain values, the same thing happened with the other FA.

In consequence, we will provide detailed simulation results only for the 'Crafter' FA in this paper.

First a reset signal is applied to the circuit and it goes to the initial state, then we applied two types of inputs:

1) when the degrees of membership of the two inputs are equal and their values first decrease from 1.0 till 0.1 in steps of 0.1, then increase again to 1.0 and finally decrease to 0.1, also in steps of 0.1.
2) when one input has first a degree of membership of 0.1 and the degree of membership of the other input decreases from 1.0 to 0.1 and goes back to 1.0, all in steps of 0.1; then the first value changes directly to 0.9 and the second input repeats its behaviour.

No methods for adjusting the degree of membership (i.e. conservation or normalization of states) are applied in the first and the second sets of simulations (subsection IV-B), while in subsection IV-C we apply either conservation of state, or normalization of states.

On all the following diagrams the horizontal axis contains the simulation time, in clock cycles (the state changes only at the rising edge of the clock signal), while on the vertical axis we represent the degree of membership of the inputs and of the states of the FA. The degrees of membership are normalized to integer values between 0 and 100 instead of real number in the interval $[0, 1]$. The inputs are represented with dashed lines, and the states with continuous lines.

B. First problem: controllability of FA

From the four pairs of s- and t-norms used for the functions $F2$ and $F1$, only the algebraic FA proved to be controllable. For the max-min FA, the degree of membership of its states took the smallest value of the degree of membership of its inputs (0.1 in this case) and after that their values could not be increased, while for the bounded and drastic FA, the degrees of membership of all their states went to zero and remained zero after that, no matter what happened at inputs. In consequence we decided to try to use different s-norms for function $F2$, but only algebraic product for $F1$, and the following diagrams are for these cases.

The following figures are for the first case (equal inputs). Figure 1 shows the results for algebraic sum and product FA, and figure 2 for the max-product functions.

It can be observed that the max-product automaton has an undesirable behaviour: when the degree of membership of its inputs decreases so does the degree of membership of its states (of all of them!), but after that, when the degree of membership increases again, the states remain with a zero degree of membership.

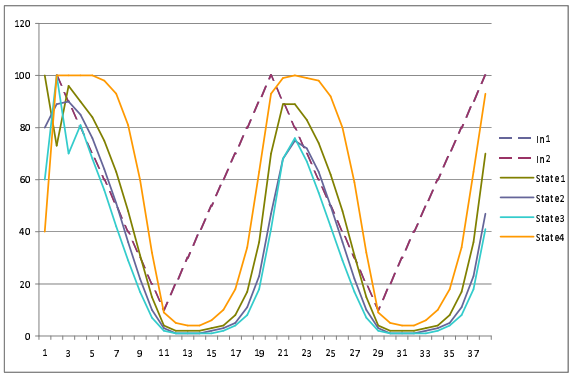

Fig. 1. Algebraic FA with equal inputs and without conservation of state

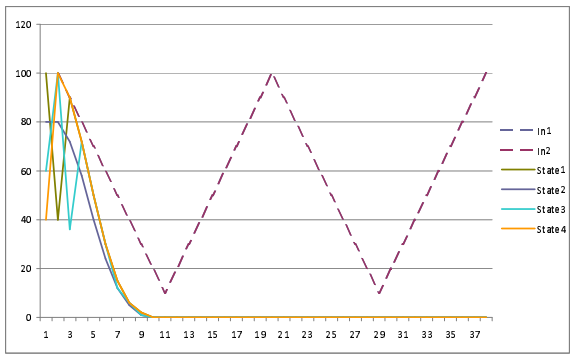

Fig. 2. Max-product FA with equal inputs and without conservation of state

The algebraic automaton looks more controllable, in the sense that when the membership values of the inputs decrease so does the degree of membership for states, but when the degree of membership of inputs increase again, the degree of membership of the states follows it. This means that the inputs can control the states. It may seem a bit unusual that the degree of membership of all of its states goes towards 1.0, but this is a behaviour that can be expected from a fuzzy automaton.

The automaton with drastic sum and algebraic product is not represented because the degree of membership of its states is 1.0 almost all the time, which is also true for the next simulation scenarios, which makes this type of automata unusable.

The automata with bounded sum and algebraic product used for $F2$ and $F1$ and the FA with Einstein sum for $F2$ and algebraic product for $F1$ behave very much the same like the FA with algebraic sum and algebraic product not only in this simulation scenario, but also in the other scenarios that we simulated. Hence, they will not be presented further. (For Einstein sum see [18], chapter 3, section 3.2.2, page 32.)

In the second simulation scenario one input has either a low, or a high degree of membership, while the other input changes like in the first scenario. Figure 3 is for the max-product automaton and figure 4 is for the algebraic automaton. Again, the degree of membership of the states of the max-product automaton go to zero and do not increase after that, while the algebraic automaton has a better behaviour, in the sense that its states can be influenced by the inputs.

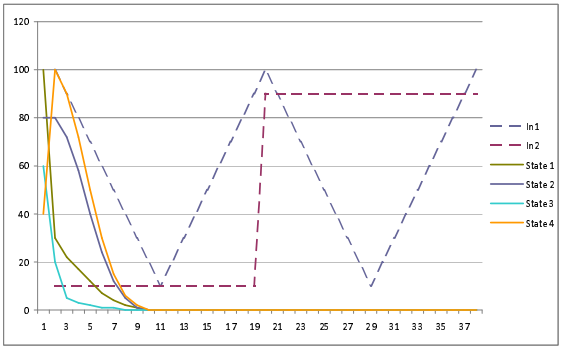

Fig. 3. Max-product FA without conservation of state, non-equal inputs

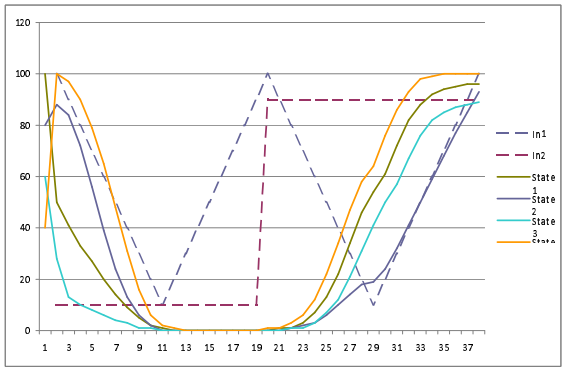

Fig. 4. Algebraic FA without conservation of state, non-equal inputs

C. Methods used to improve FA's behaviour

In [15] the authors propose the method called *conservation of state*, in order to overcome the problem of the continuous decrease of the degree of membership of the fuzzy states. However, in [15] the method is illustrated only for a small automaton, with only two states. We apply it to the 'Crafter' automaton with equal inputs (like in scenario 1) and non-equal inputs (scenario 2), with all the s- and t-norms described in subsection IV-B.

Figures 5 and 6 contain the results for the max-product automaton. It can be seen that, although it takes longer than in the case without conservation of states till the degrees of membership of the states go to zero, this thing eventually happens, and after the states have a zero degree, their degrees cannot be increased further.

As can be seen from figures 7 and 8, the unpleasant surprise is that, in what the algebraic automata are concerned, the method of conservation of state worsens their behaviour: the degrees of membership of their states remain very high, mostly between 0.8 and 1.0. This behaviour can be explained by the contribution given by the negated values of the degrees of the inputs: when the degree of an input is low, its negation (i.e

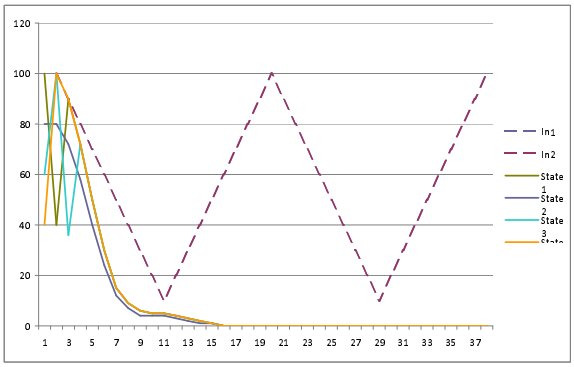

Fig. 5. Max-product FA with conservation of state, equal inputs

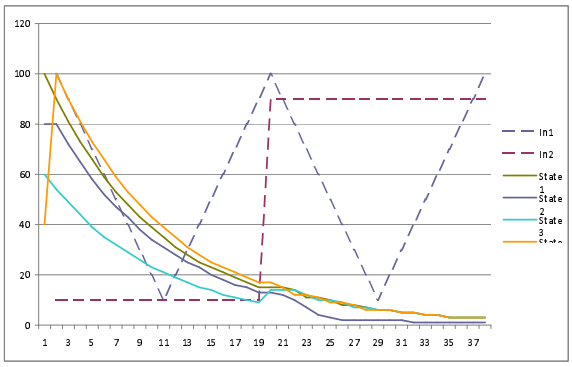

Fig. 6. Max-product FA with conservation of state, non-equal inputs

$1 - degree$) is high, and the consequence is that the degrees of states remain high.

We illustrate this behaviour only for the algebraic automaton in the two cases mentioned above (equal and non-equal inputs), but the FA with bounded sum and algebraic product and FA with Einstein sum and algebraic product have a very similar behaviour. Also, the negative effects of conservation of states has been observed on the FA from [7], example 1.

We can conclude that the conservation of state method is not efficient, at least not for the studied automata.

In [16], Watanebe et al propose to *normalize the states* of the

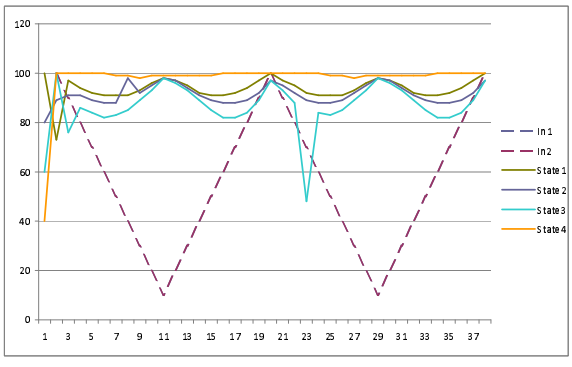

Fig. 7. Algebraic FA with conservation of state, equal inputs

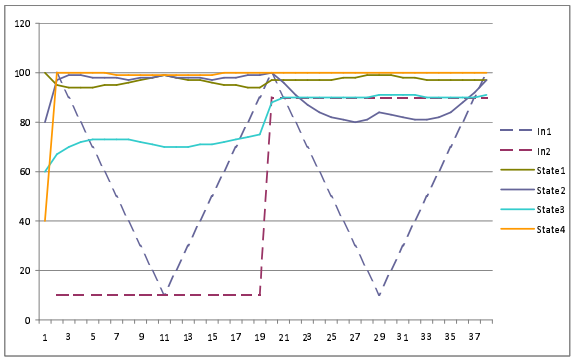

Fig. 8. Algebraic FA with conservation of state, non-equal inputs

fuzzy automata in order to avoid their degrees to decrease to zero. This means to determine the maximum value among the degree of membership of all states and to divide all the degrees of membership of the states to that maximum. In this way obviously at least one state will have the value 1 for its degree of membership. We will present the effect of this method on the max-product automaton. We apply it only to max-product automaton because its states will go to zero otherwise. Since the algebraic automata behave well in the studied example, we do not need to apply this computationally intensive method to them. We consider that the method of state normalization is too computationally intensive for an implementation in hardware, because it involves division operations.

Figures 9 and 10 contain the results for equal and non-equal inputs. In the first case the states will have very soon the maximum degree of membership and will remain unchanged, while in the second case they will have also a high degree of membership (between 0.9 and 1.0), with the exception of the state 2.

In the 'Crafter' example the method of state normalization proves to be inefficient.

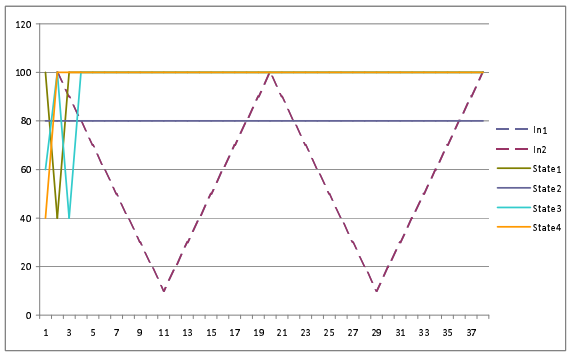

Fig. 9. Max-product FA without conservation of state and with state normalization, equal inputs

V. Conclusions and Future Work

In this work we have implemented a VHDL framework for the study of fuzzy automata. Our framework 'unifies' the approaches presented in [15], [19] and [7]. The implementation

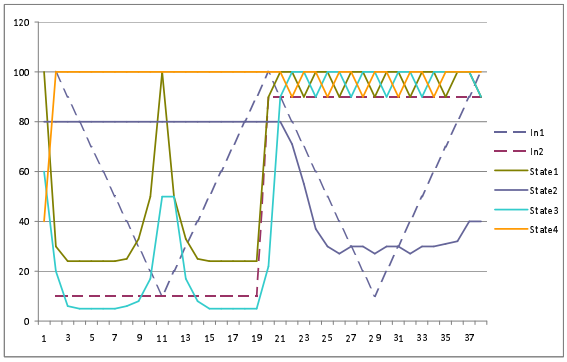

Fig. 10. Max-product FA without conservation of state and with state normalization, non-equal inputs

is flexible and, because of the parametrization of FA, it is also efficient in performing many simulations in an easy way (even in parallel).

The framework permits to test the efficiency of different functions used for state transitions of FA, as shown by our simulation results. Our simulations suggests that the maximum and minimum functions (the most used in the literature) do not allow the state of the automata to be controlled by its inputs. On the other side, the algebraic sum and product, the bounded sum and algebraic product, and the combination of Einstein sum and algebraic product proved to be efficient.

We implemented two methods proposed in the literature in order to avoid the degrees of membership of the states of FA to go to zero: the conservation of state (from [15]), and the state normalization from [16]. In our simulations both methods gave bad results, which is a new result.

The flexibility of our framework will allow us to extend our investigations on the performance of fuzzy automata and to do the following tasks:

- to perform simulations and performance studies on other examples of FA, in order to obtain general conclusions, e.g., which functions are more suited for implementing state transitions.
- to include in our framework other types of FA (e.g. FA with fuzzy relief, Chen's GFA, etc).
- to perform FPGA implementation (i.e. synthesis) of several types of FA. The synthesis will allow us to make a more realistic evaluation of resources used for different types of FA. For example, a state normalization operation used in order to maintain a non-zero state of FA can be very costly in a hardware implementation because it involves a division operation. Also, the Einstein sum can be resource consuming because it uses the division operation as well.
- to find some interesting applications for FA. Most often the application from the current literature are either very small examples (toy applications, e.g. [13]), or they are extremely complex systems, like the hybrid fuzzy-crisp automata from the works of Grantner and Fodor ([22]).

REFERENCES

[1] A. Stamenković and M. Ćirić, "Construction of fuzzy automata from fuzzy regular expressions," *Fuzzy Sets and Systems*, vol. 199, pp. 1–27, July 2012.

[2] M. Ćirić, J. Ignjatović, N. Damljanović, and M. Basić, "Bisimulations for fuzzy automata," *Fuzzy Sets and Systems*, vol. 186, no. 1, pp. 100–139, January 2012.

[3] Y. Cao and Y. Ezawa, "Nondeterministic fuzzy automata," *Information Sciences*, vol. 191, pp. 86–97, May 2012.

[4] K. W. Schmidt and Y. Boutalis, "Fuzzy discrete event systems for multi-objective control: Framework and application to mobile robot navigation," *IEEE Transactions on Fuzzy Systems, accepted for publication*, 2012.

[5] Q.-E. Wua, X.-M. Pangc, and Z.-Y. Hana, "Fuzzy automata system with application to target recognition based on image processing," *Computers & Mathematics with Applications*, vol. 61, no. 5, pp. 1267–1277, March 2011.

[6] G. Bailador and G. Triviño, "Pattern recognition using temporal fuzzy automata," *Fuzzy Sets and Systems*, vol. 161, pp. 37–55, 2010.

[7] M. Doostfatemeh and S. Kremer, "New directions in fuzzy automata," *International Journal of Approximate Reasoning*, vol. 38, pp. 175–214, 2005.

[8] A. Wielgus and M. Maciag, "Digital implementation of a programmable reconfigurable fuzzy automaton for control applications," in *14th International Conference MIXDES 2007*, Chiechocinek, Poland, June 21-23 2007, pp. 270–273.

[9] C.-L. Chen, "Programmable fuzzy logic device for sequential fuzzy logic synthesis," in *IEEE International Fuzzy System Conference*, Melbourne, Australia, 2001, pp. 107–110.

[10] ——, "Generalized fuzzy automata for fuzzy feedback control with words," in *Proceedings of 2004 International Computer Symposium ICS2004*, Taiwan, December 15-17 2004, pp. 895–900.

[11] C.-L. Chen and P.-C. Hsiao, "Supporting QoS in wireless MAC by fuzzy logic control," in *Proceedings of IEEE Wireless Communications and Networking Conference WCNC'05*, vol. 2, New Orleans, USA, March 2005, pp. 1242–1247.

[12] C.-L. Chen, H.-L. Yang, and Y.-D. Huang, "A cross-layer control based on fuzzy automata for ad-hoc WLAN QoS," in *Proceedings of the 2006 Joint Conference on Information Science JCIS 2006*, Kaohsiung, Taiwan, October, 8-11 2006.

[13] J. Virant and N. Zimic, "Fuzzy automata with fuzzy relief," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 1, pp. 69–74, february 1995.

[14] F. Steimann and K.-P. Adlassnig, "Clinical monitoring with fuzzy automata," *Fuzzy Sets and Systems*, vol. 61, pp. 37–42, 1994.

[15] Y. Mori, K. Otsuka, and M. Mukaidono, "Properties of fuzzy sequential circuit using fuzzy transition matrix and their design method," in *Fuzzy System 1995, Proceedings of 1995 IEEE International Conference on*, vol. 4, Yokohama, Mar. 20–24, 1995, pp. 2133–2138.

[16] T. Watanabe, M. Matsumoto, and M. Enokida, "Synthesis of synchronous fuzzy sequential circuits," in *Proceedings of third IFSA Word Congress*, 1989, pp. 288–291.

[17] J. Virant, N. Zimic, and M. Mraz, "Fuzzy sequential circuits and automata," in *Fuzzy Theory Systems: Techniques and Applications, vol. IV*, C. T. Leondes, Ed. Academic Press Inc., 1999.

[18] H.-J. Zimmermann, *Fuzzy Set Theory – and Its Applications, Second, Revised Edition*. Kluwer Academic Publishers, 1991.

[19] F. M. McNeill. The intelligence crafter: a fuzzy state machine builder program. Fuzzy Systems Engineering 619-748-7384. [Online]. Available: http://www.foretrade.com/Documents/fuzautom.pdf

[20] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, 1995.

[21] J. Virant and N. Zimic, "Attention to time in fuzzy logic," *Fuzzy Sets and Systems*, vol. 82, pp. 39–49, february 1996.

[22] J. Grantner and G. Fodor, "Fuzzy automata for intelligent hybrid control systems," in *Proceedings of the 2002 IEEE International Conference FUZZ-IEEE'02*, vol. 2, May 12-17 2002, pp. 1027–1032.