8-2005

Design of a Reconfigurable State Transition Algorithm for Fuzzy Automata

Paolo A. Tamayo
Western Michigan University

DESIGN OF A RECONFIGURABLE STATE TRANSITION ALGORITHM FOR FUZZY AUTOMATA

by

Paolo A. Tamayo

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science in Engineering (Computer)
Department of Electrical and Computer Engineering

Western Michigan University
Kalamazoo, Michigan
August 2005

DESIGN OF A RECONFIGURABLE STATE TRANSITION ALGORITHM FOR FUZZY AUTOMATA

Paolo A. Tamayo, M.S.E.

Western Michigan University, 2005

An Ontological Controller, a type of supervisory controller used for complex industrial systems, is usually characterized by a large global state set. Monitoring the global state set of these systems is impractical when the desired goal path is chosen since the decision on the next state depends only on a small section of the total state space. The approach is to operate only in a small moving window of the large state transition path. The Hybrid Fuzzy-Boolean Finite State Machine (HFB-FSM) is part of the solution to implement this scheme. The HFB-FSM models the state graph along with plant data and advises the ontological controller with respect to recovery from faults. The use of reconfigurable hardware is an attractive implementation for this approach. When specified properly, reconfigurable implementation can dramatically reduce the physical hardware in the realization while maintaining the computing power of the design. The objective is to model the ever-changing state graph configurations implementing only the relevant states that affect the state transitions. The author proposes a reconfigurable architecture that will support the remodeling of the state graph. This will allow the supervisory controller to reset the fuzzy automaton in order to model a particular state cluster, as needed. The use of parameterized components in the design provides for flexibility with respect to choosing the number of fuzzy inputs and number of fuzzy states.

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude and appreciation to all the persons that made this thesis possible. Special thanks to Dr. Janos Grantner and Mr. Rama Gottipatti for giving me the opportunity to participate in their research areas and supporting me throughout the process. Special thanks are also due to Dr. Bradley Bazuin and Dr. Liang Dong for being my thesis advisers.

I also would like to acknowledge the faculty and staff of the Department of Electrical and Computer Engineering for their unwavering support especially to Mr. Dave Florida for the tools and licenses used. Special thanks to my family and friends for their inspiration and support in finishing my studies. Finally, my sincere thanks to my lovely wife Agnes and my wonderful son Gabriel for their love and understanding in this endeavor.

Paolo A. Tamayo

TABLE OF CONTENTS

Table of Contents—Continued

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Current trends in computing are targeting the costs of development and implementation. Due to these reasons we see the emergence of reusable design and reconfigurable designs. The use of reconfigurable hardware is indeed an attractive feature in any systems since it can dramatically reduce the physical hardware required in an implementation while maintaining the computing power of the design when specified properly. If reconfiguration can be accomplished without affecting the functions and performance of the system, the hardware implementation can support functions that would otherwise require bigger hardware components than the actual hardware implemented. This scenario is often seen in large scale, complex industrial control settings. Controllers are usually composed of a two-level structure: a plant control level and a supervisory level. From the name itself, the plant control level is the control system that handles the functions of the plant. The supervisory control on the other hand deals with the global state space of the whole system. The supervisory control tracks the operation of the plant controls. Monitoring the operation at this level is impractical if the global state space is huge. It was shown by Fodor [5] that monitoring a small section of the global state space is sufficient to determine the next state of the control path if the current goal state is known. The idea here is to create a moving window over the state transition graph of the system and monitor the control actions in that window. The path and states that are enclosed in the window is the current active path and states in the operation [7].

Ontological Control (a special type of supervisory control) is concerned with the automatic detection of faults based on the violations of the ontological assumptions. Previous literature [9] describes the ontological control to have a deterministic operation, i.e. a single state can be selected in any situation. Ontological control makes sure that

there will always be a next state that will be selected. The basis of the choice of the next state is dependent on the defined goal path of the plant being controlled. In recent research it has been proposed that the controller is enhanced with the Fuzzy-State-Fuzzy-Output Finite State Machine (FSFO FSM) [8] to decide on the next state. Either this next state will have the appropriate actions to recover from the fault and keep the plant on a suitable goal path or it will indicate a non-recoverable fault to the controller.

The FSFO FSM model was extended to form the Hybrid Fuzzy-Boolean Finite State Machine (HFB FSM) [3]. This automata design addresses the ontological control functions of error detection and recovery. Since the intended applications of the HFB FSM involves a large state set, another recent research suggests an approach using the property of virtual fuzzy automata to manage the state supervision [10]. This solution, based on the HFB FSM model, outlines the creation of the automata to handle the next state determination based on the current state and the conditions of the inputs. The virtual fuzzy automata instance is created and information on the state transition graph, conditions for transition, initial state, fuzzy inputs and other relevant information is downloaded. The determined next state is then passed to the supervisory controller for processing. This research is based upon the virtual fuzzy automata approach to implement the HFB FSM model. The author aims to establish the framework for the creation of the virtual fuzzy automaton into hardware. Since reconfiguration of the virtual fuzzy automata is frequent in the operation of the Ontological Control, the use of reconfigurable architecture realizes a more maneagable implementation of the automata into hardware. This research proposes an architecture to implement such reconfigurable system. The proposed architecture implements a configuration for Multiple-Input-Single-Output (MISO) systems. It will also serve as a generic design model to implement Multiple-Input-Multiple-Ouput (MIMO) systems.

The objective of this research is summarized as follows:

1. Design the architecture of a reconfigurable state transition algorithm for a fuzzy automata.
2. Systematically establish the framework for a virtual fuzzy automaton.

To map the reconfigurable design into hardware parameterized components are utilized. The design is implemented using, the industry standard, Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL).

Chapter 2 decribes the HFB FSM Model components and the B Algorithm [1]. A simple example on model building and inference operation is also shown. The B Algorithm is the heart of the implementation of the HFB FSM. The B Algorithm is the operation where the actual mapping of a fuzzy input value to a unique set of Boolean variables is performed to determine the next state of the HFB FSM.

Chapter 3 describes the detailed design of the architecture. That includes the partitioning of the design modules and the parameterized components that implement the reconfigurable characteristics of the design. Since the design is actually modular and can easily be integrated into a larger system, a section outlines the interfacing requirements.

Chapter 4 describes the testing and verification methodology of the design. A previously developed software implementation of the HFB-FSM [6] is used to check the hardware design simulation results. The same set of parameters are applied to both design and the results are compared. Both results are expected to match.

Conclusion are discussed in Chapter 5. An outline and brief description of some possible future research directions are also given.

The Appendix shows the design files and the simulation files used in verification.

CHAPTER 2

REVIEW OF LITERATURE

2.1 Ontological Control

Ontological Control is a contemporary supervisory control approach. It concerns about automated fault detection and identification of some particular types of faults referred to as ontological desynchronization, and the recovery from those faults in large, complex industrial control systems. When the plant that is controlled is well defined, there is always a corresponding control action on every state or pre-determined condition. If some external factors introduces a new state (not part of the pre-determined conditions) to the plant, the control action may result in three possible scenarios to occur [9]. Case 1: The resulting control action yields a transition into one of the predefined states; Case 2: The resulting control action results in an expected modification of the plant and ; Case 3: The plant changes in an unexpected way. The latter one is the desynchronization case. Ontological control must be able to launch a synchronization procedure to bring back the plant to one of the pre-defined goal paths. Sources of desynchronization include unexpected external actions, incomplete plant definition (i.e. some states are not defined properly) and violations of the ontological assumptions. Ontological assumptions are the ones that generate the desired plant outputs. However, these assumptions form the cradle of the control actions that the system does not check to see if they are valid.

It has been proposed in previous literature that the Hybrid Fuzzy-Boolean Finite State Machine (HFB FSM) can be used to address the problem of recovery from ontological desynchronization [2][3]. Ontological Controller (OC) uses the HFB FSM to do a recovery when unexpected changes or faults occur in the operation. In other words, the HFB FSM is used to perform recovery operations on another controller when the recovery specifications of that controller are known. The general idea is that the OC uses the HFB

FSM to model the relevant section of the control algorithm along with plant data to make a decision on the next state of the goal path. In order to do that the Ontological Controller needs to reconfigure the HFB FSM for each of those particular state clusters of the control algorithm and then pass the actual plant data to the HFB FSM. If there is a fault, the HFB FSM will detect it through a state transition and advise the OC whether a recovery is possible from the fault. Since the configurations of the critical state clusters with respect to the ontological desynchronization fault keep changing along the goal path, the OC needs to reconfigure the HFB FSM for every state cluster of concern.

2.2 The HFB-FSM Model

The HFB FSM model was first introduced in [3] for ontological control. To address additional requirements in modeling complex hybrid systems it was extended in [2] (see Figure 2.1).

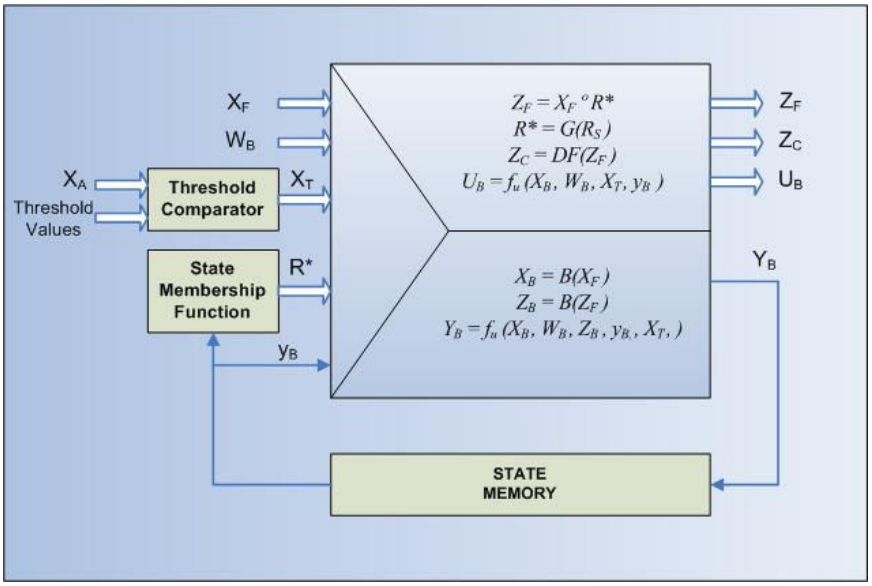

Figure 2.1 Extended HFB-FSM Model [2]

The HFB FSM model is implemented by a Boolean automaton based upon two valued logic. The model is defined by the following formulas:

$$Z_F = X_F \; o \; R^* \tag{2.1}$$

$$R^* = G(R_S) \tag{2.2}$$

$$Z_C = DF(Z_F) \tag{2.3}$$

$$U_B = f_u (X_B, W_B, X_T, y_B) \tag{2.4}$$

$$X_B = B(X_F) \tag{2.5}$$

$$Z_B = B(Z_F) \tag{2.6}$$

$$Y_B = f_u (X_B, W_B, Z_B, y_B, X_T,) \tag{2.7}$$

X_F, W_B and X_A stand for fuzzy, 2-valued (Boolean), and analog inputs with associated threshold values, respectively. X_T is the result of the comparison of the analog inputs with the associated threshold values. Z_F, Z_C and U_B stand for fuzzy, defuzzified fuzzy, and two-valued (Boolean) outputs, respectively. R* is the composite linguistic model [4], and ◦ is the operator of composition.

A fuzzy state is made up of a set of crisp states, the HFB FSM stays simultaneously in all of them, to a certain degree in each. There is a dominant crisp state in this state set for which the degree of the state membership function is 1. Therefore, a fuzzy state is defined by its dominant crisp (Boolean) state and a state membership function

$$S_{Fk} : S_k , g_{Sk} \tag{2.8}$$

where S_{Fk} stands for fuzzy state k, S_k represents crisp state k and g_{Sk} is the state membership function associated with S_k.

Each crisp state of the HFB FSM is characterized by an overall linguistic model R_S, or by a set of linguistic sub-models in the case of multiple-input-single-output (MISO), and multiple-input-multiple-output (MIMO) systems [2].

In (Equation 2.2), G stands for the matrix of state membership functions, where

$$G = \begin{bmatrix} \beta_1^1 & \beta_2^1 & \cdots & \beta_p^1 \\ \beta_1^2 & \beta_2^2 & \cdots & \beta_p^2 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_1^p & \beta_2^p & \cdots & \beta_p^p \end{bmatrix} \text{ or } G = \begin{bmatrix} g_{s1} \\ g_{s2} \\ \vdots \\ g_{sp} \end{bmatrix} \tag{2.9}$$

For each fuzzy state of the HFB FSM model, an R* composite linguistic model is created from the finite set of R_{Si} overall linguistic models (i = 1, …,p). Let the HFB FSM be in a fuzzy state R_{Fk}, then

$$R^*_k = max[min(\beta_1^k, R_{S1}), \ldots, min(\beta_p^k, R_{Sp})] \tag{2.10}$$

where β_1^k β_p^k stand for the degrees of state membership function g_{Sk} ,where $g_{Sk} = [\beta_1^k \ \beta_2^k \ \beta_3^k,\ldots,\beta_p^k]$ from Equation 2.9, and $R_{S1},\ldots,R_{Sp}$ are the overall linguistic models in crisp states $S_1,\ldots,S_p$, respectively. By modifying the degrees of the state membership function (β) on-line, new R* composite linguistic models can be created under real time conditions.

X_B, Z_B, Y_B and y_B stand for two-valued Boolean input, Boolean output and next state and present state of the variables, respectively. B stands for the Fuzzy-to-Boolean transformation algorithm to map a change in the status of fuzzy variable into state changes of a finite set of corresponding Boolean variables. The Z_C crisp values of the fuzzy outputs are obtained by evaluating a defuzzification strategy, DF.

This research assumes only a multiple fuzzy input and single fuzzy output implementation of the HFB FSM. The analog thresholds and digital inputs are omitted. Chapter 3.1 explains the implementation in detail. Future work and enhancement of the design can include these factors, which will be discussed in detail in Chapter 5.

2.3 B Algorithm

The Fuzzy-to-Boolean Transformation Algorithm is implemented to both inputs and outputs (Equations 2.5 and 2.6) [8]. The algorithm takes the average position of the maxima, obtained using the Mean-of-Maxima (MOM) defuzzification method, in the universal space for the input fuzzy variable or output fuzzy variable. The universal space defines all the possible elements of concern in which a particular input or output can be formed. For example, the universal space for a speed input is defined in the range of 0 to 16 miles per hour. The universal space is then divided into sub-intervals according to the number of linguistic variables defined. Figure 2.2 below shows the linguistic variables chosen for the speed. The figure also shows that the speed input shows the occurrence of a maximum membership at 0.8 in three locations. The MOM defuzzification method takes the average of the occurrence of these maxima. From the figure the computed MOM value is 9 (since {8 + 9 + 10}/3) which falls in the interval of fast speed.

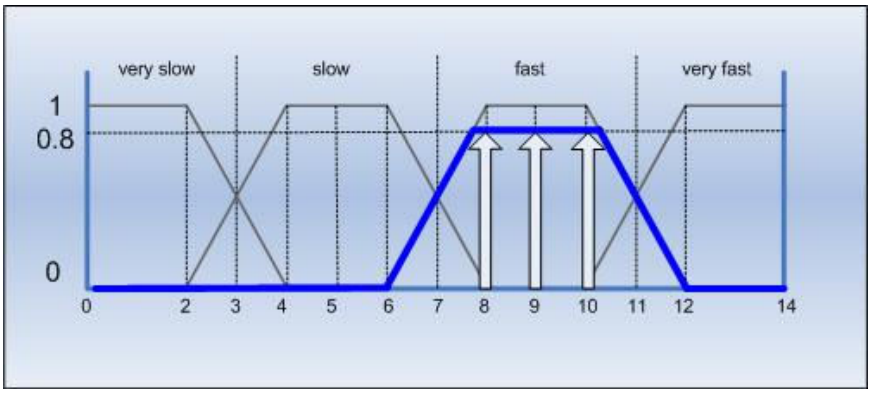

Figure 2.2 Mean-of-Maxima Calculation Example

The created sub-intervals are then assigned unique Boolean numbers. These Boolean assignments are then used to detect the current condition that particular fuzzy variable. Based on the location of the MOM we can determine the present status of the fuzzy variable. Collating the MOM of all the fuzzy variables in the system forms the overall picture of the current condition of the system This information is then used to determine

the next state transition of the automata of the system. Chapter 3.1 explains the MOM method implementation in detail. In the hardware implementation, only the inputs are used to determine the next state of the system.

2.4 Model Building and Inference Operation

Based on the linguistic model, an overall fuzzy relation of the inputs to the output for the MISO system is created. The linguistic model is the verbal description of the relations of the inputs and output. These verbal descriptions are the rules that define the behavior of the output based on the observed inputs and the current state of the system. Often times there are a number of these rules and they must be combined to form the overall rules that will describe the system. Rule composition or model building (Equation. 2.1) is the method of combining all of these rules.

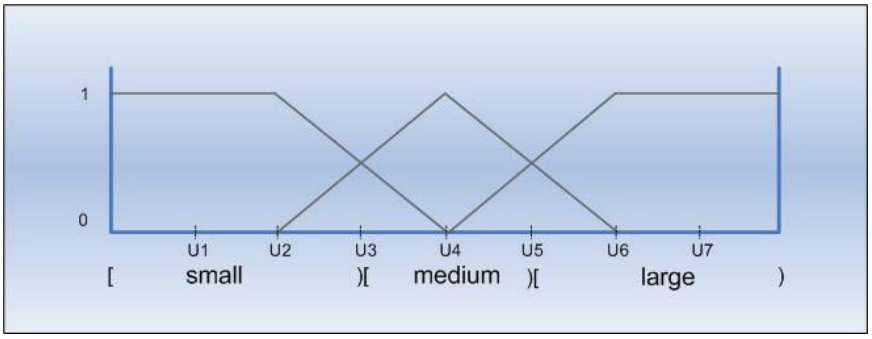

Figure 2.3 Inference Example Intervals Used

To illustrate the operation of the composition lets take a simple example. Assume a MISO system with two inputs, three states and one output. Assume that the system uses the normalized universal set for the two inputs and single output shown in Figure 2.3. The linguistic model of the system is shown in Table 2.1. Where X, Y and Z variables are the fuzzy input 1, fuzzy input 2 and fuzzy output, respectively. Graphically, the fuzzy sets are shown in Figure 2.4.

Table 2.1 Liguistic Model Example

Rules	Rule Description
For Crsip State 1	
Rule 1	If X is small and Y is small then Z is **small**.
Rule 2	If X is medium and Y is large then Z is **large**.
Rule 3	If X is large and Y is medium then Z is **medium**.
Rule 4	If X is large and Y is large then Z is **large**.
Rule 5	If X is medium and Y is medium then Z is **medium**.
For Crisp State 2	
Rule 1	If X is large and Y is small then Z is **medium**.
Rule 2	If X is small and Y is medium then Z is **small**.
Rule 3	If X is medium and Y is small then Z is **small**.
Rule 4	If X is medium and Y is large then Z is **large**.
Rule 5	If X is large and Y is large then Z is **large.**
For Crisp State 3	
Rule 1	If X is large and Y is medium then Z is **large**.
Rule 2	If X is small and Y is large then Z is **medium**.
Rule 3	If X is large and Y is small then Z is **medium**.
Rule 4	If X is small and Y is small then Z is **small**.
Rule 5	If X is medium and Y is medium then Z is **medium.**

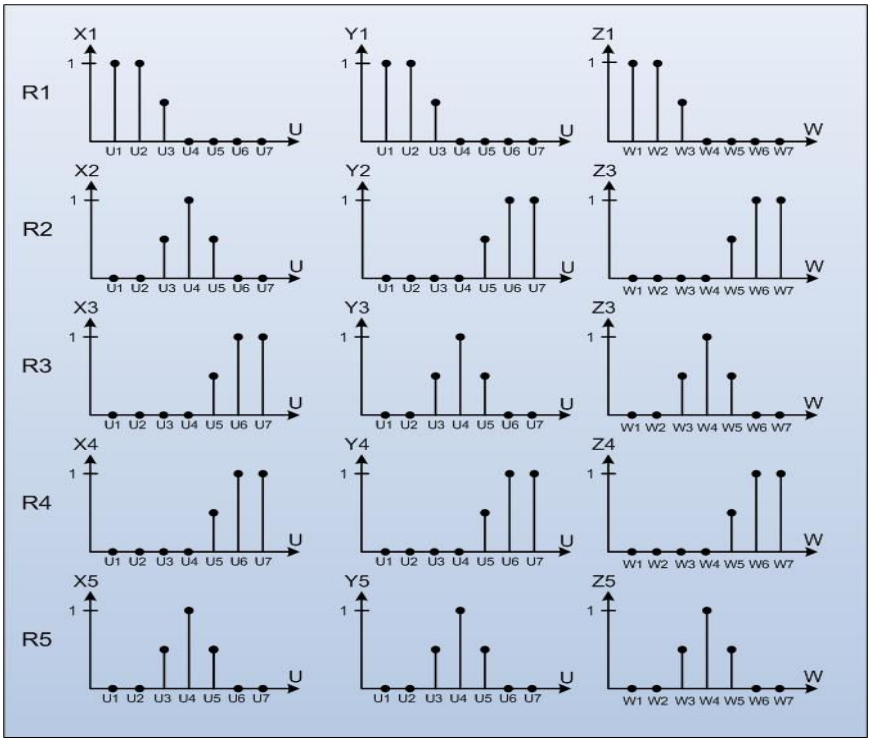

Figure 2.4 Graphical Views of Fuzzy Sets in Crisp State 1

It was shown in the work of Grantner [7] that the fuzzy relation Rx1 for X1 and Z1 is given by the formula:

$$Rx1 = X1 \times Z1 \tag{2.11}$$

$$Rx1(u,w) = min(X1(u), Z1(w)) \tag{2.12}$$

Where Z1 is the expected fuzzy output based on the observed fuzzy input X1. Similarly, generating fuzzy relation Ry1 for Y1 and Z1 is achieved by replacing the X1 with Y1 in equations 2.11 and 2.12.

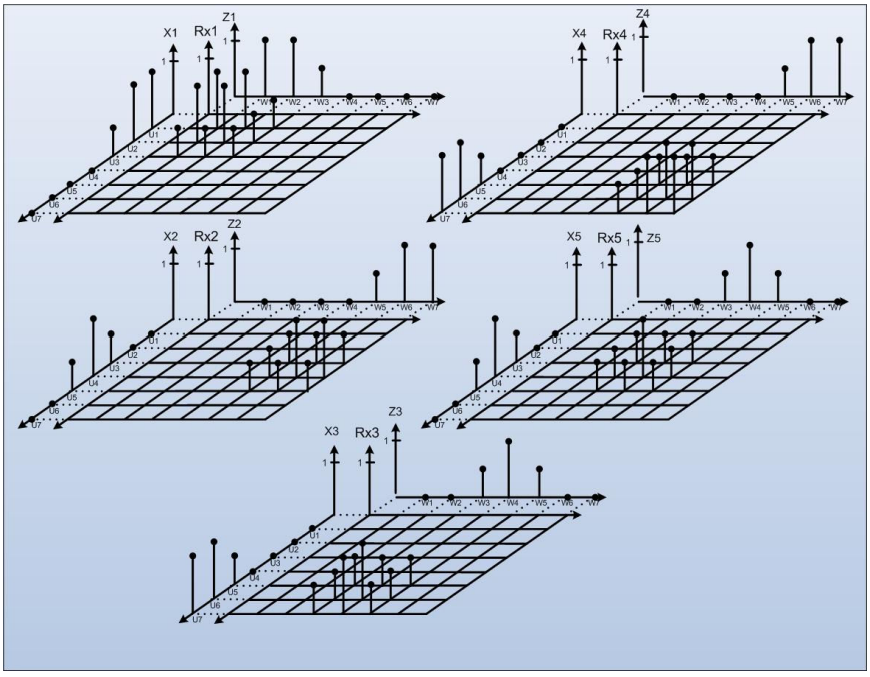

Figure 2.5 Fuzzy Relations of input X and output Z in Crisp State 1

Figure 2.5 shows the graphical representation of the fuzzy relation of input X and output Z. For each pair of Xn and Zn a relation Rn, where n is from 1 to number of elements in the universal set, is generated. To get the overall rule R (aggregated rule) for input X and output Z the equation below is used [7]:

$$Rx = Rx1 \ U \ Rx2 \ U \ Rx3 \ U \ Rx4 \ U \ Rx5 \quad (2.13)$$

$$Rx(u,w) = max(Rx1(u,w), Rx2(u,w), Rx3(u,w), Rx4(u,w), Rx5(u,w)) \quad (2.14)$$

By superimposing each of the fuzzy relation Rx1, Rx2, Rx3, Rx4 and Rx5 shown in Figure 2.5 with each other and taking the maximum value of all the points that contains a non-zero relation, Rx is calculated. The same sequence of operation is performed to derive Ry for input Y. The graphical representation of the aggregated rules Rx and Ry are shown in Figure 2.6. Since in the system there are three crisp states, two more sets of aggregated rules (one pair for each remaining state) are generated by the remaining fuzzy relations given in Table 2.1.

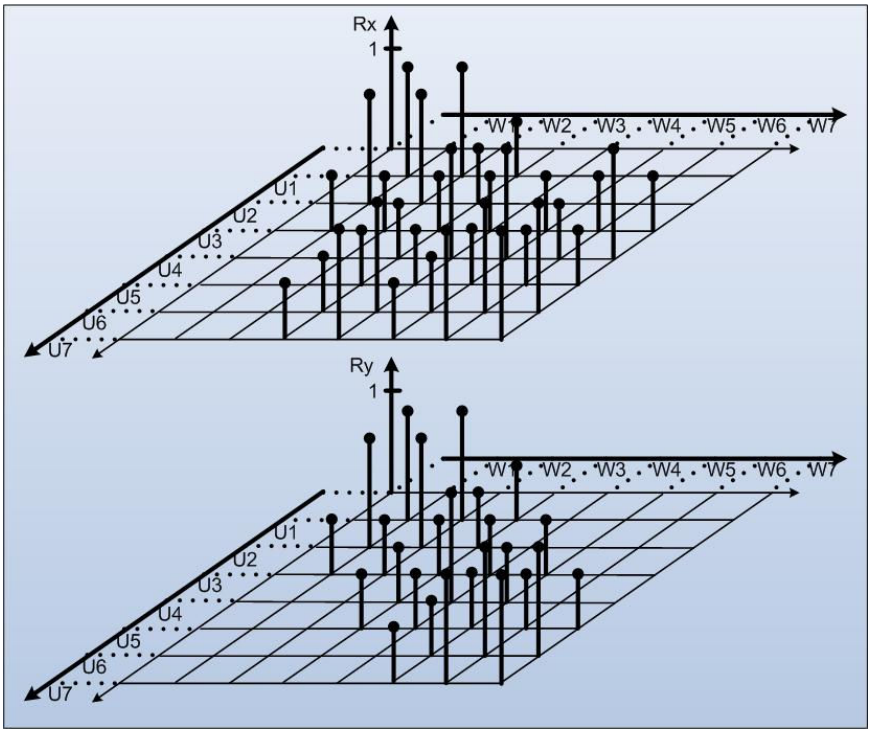

Figure 2.6 Aggregated Rules Rx and Ry in Crisp State 1

After the aggregated rules in crisp states are generated, the fuzzy state rules or the overall linguistic model is created based on these crisp state rules. From equation 2.8, we know that a fuzzy state is defined by its dominant crisp (Boolean) state and a state membership function. To generate the overall linguistic model we need to assign the values of the state membership function g_{Sk}. A G-matrix, shown in equation 2.9, of degrees of state membership functions described in [4] is used as the g_{Sk} (where k ranges from 1 to p states). For the example p = 3 since there are three states. β^i_k here denotes the degree of state membership to state S_k. Mathematically, the composite overall linguistic model is defined by equation 2.10.

The choice of the values of β^i_k is dependent on the modeling of the system behavior. The G matrix actually provides a means for the system designer to tune the outputs by manipulating the elements of the matrix. This can be shown later in inference that the fuzzy outputs are generated using the overall linguistic model R^*_k derived from the G matrix and the crisp state rules.

For our example, the chosen values for the elements in the G matrix are shown below:

$$G = \begin{bmatrix} 1.0 & 0.6 & 0.3 \\ 0.4 & 1.0 & 0.7 \\ 0.5 & 0.2 & 1.0 \end{bmatrix}$$

Currently, the choice of values is dependent on the specification of the system and the experience of the system designers. As of this writing, there is no formal method of determining the optimum values for the elements in the G matrix. Notice that the diagonals are all 1.0. This simply means that there is always one dominant state in any fuzzy states. All the other states active on that particular fuzzy state will have a degree of membership defined by their corresponding values in the G matrix. .For example in row 1,

state 1 will be dominant and state 2 and 3 will have their degree of membership at 0.6 and 0.3, respectively.

To calculate the final fuzzy state rules for each fuzzy input in each fuzzy state, we simply perform the operation of Equation 2.9. For the example given, the computations are show below:

Calculation of the crisp state rules for fuzzy input x is shown below:

$$Rx_1 = \begin{bmatrix} 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \end{bmatrix}, \quad Rx_2 = \begin{bmatrix} 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1.0 & 1.0 & 0.5 & 0.0 & 0.5 & 1.0 & 1.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \end{bmatrix}$$

$$Rx_3 = \begin{bmatrix} 1.0 & 1.0 & 0.5 & 1.0 & 0.5 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.5 & 1.0 & 0.5 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \end{bmatrix}$$

Where: Rx_1, Rx_2, and Rx_3 are for crisp state 1, crisp state 2 and crisp state 3, respectively. The fuzzy state 1 aggregated rule for fuzzy input x is computed as:

$$\begin{aligned} R * x_1 = \max[&\{\min[Rx_1, 1.0], \min[Rx_1, 0.6], \min[Rx_1, 0.3]\}, \\ &\{\min[Rx_2, 1.0], \min[Rx_2, 0.6], \min[Rx_3, 0.3]\}, \\ &\{\min[Rx_3, 1.0], \min[Rx_3, 0.6], \min[Rx_3, 0.3]\}] \end{aligned}$$

$$R * x_1 = \max\left\{\begin{bmatrix} 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \end{bmatrix}, \begin{bmatrix} 0.6 & 0.6 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.6 & 0.6 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \\ 0.0 & 0.0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \end{bmatrix}, \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.3 & 0.3 & 0.3 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{bmatrix},$$

$$\begin{bmatrix} 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.0 & 0.5 & 0.5 & 0.5 \\ 1.0 & 1.0 & 0.5 & 0.0 & 0.5 & 1.0 & 1.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \end{bmatrix}, \begin{bmatrix} 0.6 & 0.6 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.6 & 0.6 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.0 & 0.5 & 0.5 & 0.5 \\ 0.6 & 0.6 & 0.5 & 0.0 & 0.5 & 0.6 & 0.6 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \end{bmatrix}, \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.3 & 0.3 & 0.3 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.3 & 0.3 & 0.3 & 0.0 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.0 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{bmatrix},$$

$$\begin{bmatrix} 1.0 & 1.0 & 0.5 & 1.0 & 0.5 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.5 & 1.0 & 0.5 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \end{bmatrix}, \begin{bmatrix} 0.6 & 0.6 & 0.5 & 0.6 & 0.5 & 0.0 & 0.0 \\ 0.6 & 0.6 & 0.5 & 0.6 & 0.5 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \\ 0.0 & 0.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \end{bmatrix}, \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.0 & 0.0 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.0 & 0.0 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{bmatrix}\right\}$$

$$R * x_1 = \begin{bmatrix} 1.0 & 1.0 & 0.5 & 0.3 & 0.3 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.5 & 0.3 & 0.3 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.6 & 0.6 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.5 & 1.0 & 0.5 & 1.0 & 1.0 \end{bmatrix}$$

In addition, for fuzzy state 1 aggregated rule for fuzzy input y is found to be:

$$R * y_1 = \begin{bmatrix} 1.0 & 1.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \\ 1.0 & 1.0 & 0.5 & 0.6 & 0.5 & 0.6 & 0.6 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.6 & 0.6 & 0.5 & 1.0 & 0.5 & 0.3 & 0.3 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.5 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.3 & 0.3 & 0.5 & 1.0 & 1.0 \end{bmatrix}$$

The same sequence of operation will be performed when calculating the fuzzy state 2 and 3 aggregated rule. These rules (2 for each fuzzy state) will be the one used in the inference operation to determine the fuzzy output based on the observed fuzzy inputs. See Appendix A for the detailed summary of the generated crisp state rules and the fuzzy state rules.

To complete the example, the inference operation is shown next. Assume that fuzzy input x and fuzzy input y observed the following values:

$$X = 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.5 \quad 1.0 \quad 1.0$$

$$Y = 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.5 \quad 1.0 \quad 1.0$$

The inferred value of the fuzzy output Z is computed using Equation 2.1. Where R* is the aggregated fuzzy rule and X_F is the observed fuzzy input. This form is for the SISO type of system.

For inference in the MISO system, we simply extend Equation 2.1 to:

$$Z = min\{(X_{F1} \; o \; R^*), (X_{F2} \; o \; R^*), \ldots, (X_{Fn} \; o \; R^*)) \tag{2.15}$$

Where n is the number of inputs. Rewriting Equation 2.15 and changing X_F inputs to X and Y to represent the two fuzzy inputs we get the following equations:

$$\begin{aligned} Zx(w) = max\{&min[min(X1(u),R^*_{x11}(u,w)), min(X2(u),R^*_{x12}(u,w)), \\ &min(X3(u),R^*_{x13}(u,w)), min(X4(u),R^*_{x14}(u,w)), min(X5(u),R^*_{x15}(u,w)), \\ &min(X6(u),R^*_{x16}(u,w)), min(X7(u),R^*_{x17}(u,w))]\} \end{aligned} \tag{2.16}$$

$$\begin{aligned} Zy(w) = max\{&min[min(Y1(u),R^*_{y11}(u,w)), min(Y2(u),R^*_{y12}(u,w)), \\ &min(Y3(u),R^*_{y13}(u,w)), min(Y4(u),R^*_{y14}(u,w)), min(Y5(u),R^*_{y15}(u,w)), \\ &min(Y6(u),R^*_{y16}(u,w)), min(Y7(u),R^*_{y17}(u,w))]\} \end{aligned} \tag{2.17}$$

Where, R^*_{x11} is the first row of the aggregated rule for fuzzy input X in fuzzy state 1. Figure 2.7 shows a graphical representation of the inference operation for fuzzy input X using R^*_{X1}.

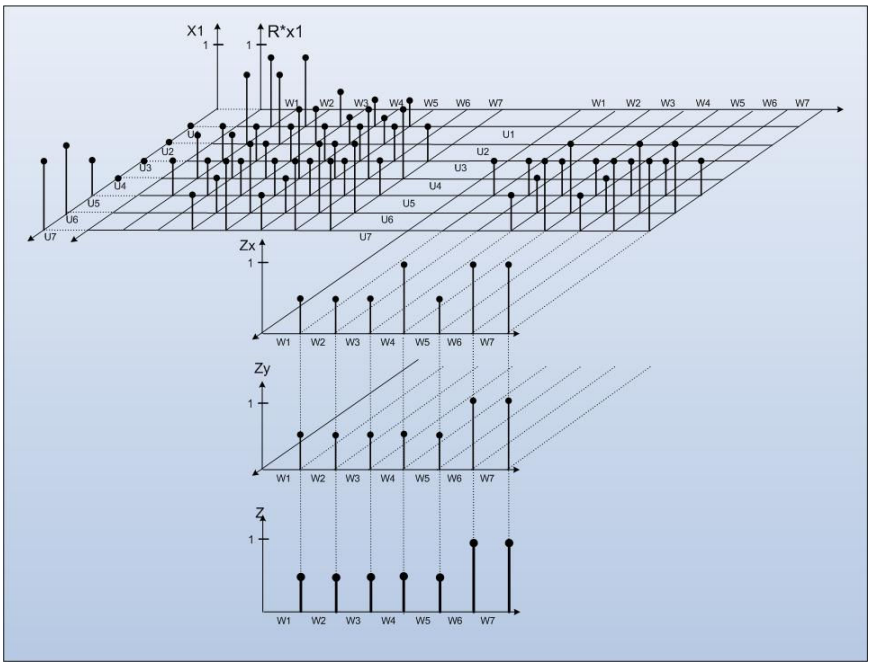

Figure 2.7 Graphical View of Fuzzy Inference

The fuzzy result Zy to fuzzy input Y1 inferred from Ry1 is shown along with the Zx result. The final step of the inference operation is the minimum operation of the two results to form the fuzzy output Z. The computation results in the Z values:

$$X = 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 1.0 \quad 1.0$$

CHAPTER 3

ARCHITECTURE DESIGN COMPONENTS

3.1 Proposed Reconfigurable Fuzzy Automata Architecture Design

The design is based on the computational model described in Chapter 2.2. The implementation is coded in VHDL. As mentioned before, the reconfigurable aspect of the design utilized parameterized components in VHDL. To design the fuzzy automata a multi-fuzzy input single output model is assumed in which the host system provides conversion of the analog inputs into its fuzzy representation. Since the design deals with the fuzzy inputs and fuzzy output only, the digital inputs and the analog inputs with threshold are omitted. This makes the design more generic since the digital inputs and the analog inputs with threshold are application specific. Equation 2.7 becomes:

$$Y_B = f_u (X_B , y_{B,}) \qquad (3.1)$$

Where X_B is the result of the generation of sets of Boolean variables for the multi-fuzzy inputs and Y_B is the result of the generation of set of Boolean variable for the fuzzy output. The generation of the Boolean variables is carried out by the B-algorithm (equations 2.5 and 2.6) which will be described in detail later. y_B is the present state and Y_B is the next state of the automata. It is shown later that the integration of n number of fuzzy inputs is easily achieved by extending the architecture of the automata.

A composition module earlier designed in [7] is included in this design to generate the fuzzy output based on the fuzzy input and the inference rule. As mentioned in chapter 2.2, the inference rule used is based on the current state of the automata. This means that for each defined state of the automata an inference rule is established based on the overall

model of the system. This operation is defined by equation 2.1 where R* is the inference rule and X_F is the fuzzy input.

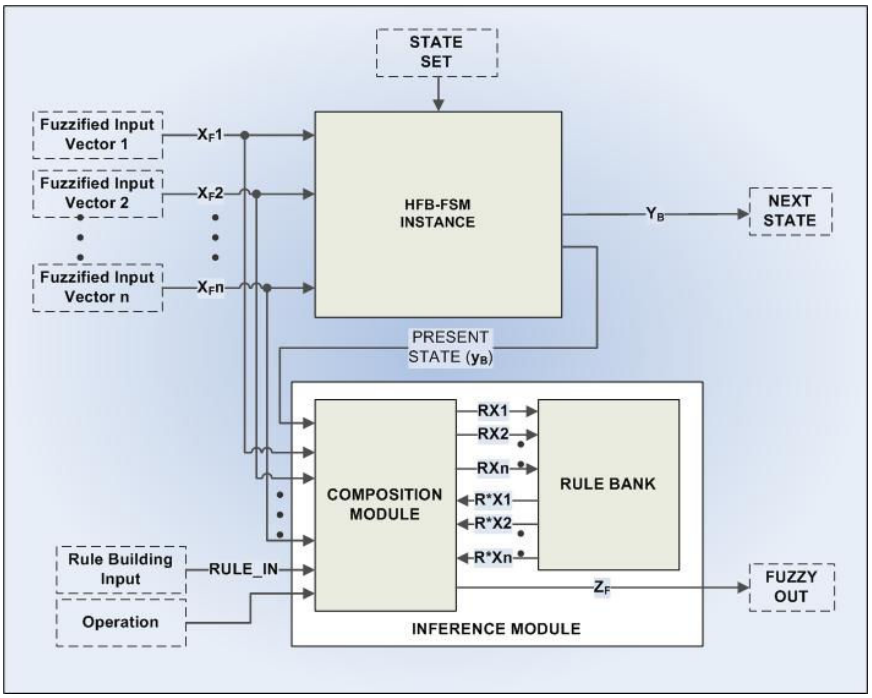

Figure 3.1 Reconfigurable Fuzzy Automata Architecture Design

Figure 3.1 shows the architecture of the reconfigurable fuzzy automata where the inputs are represented by X_Fi (where i is from 1 to n) composing the n fuzzy inputs. Each fuzzy input consists of an array of fuzzified values. Thus each input is actually a fuzzified input vector. Another input vector (RULE_IN) is provided for rule building operation for the inference module. The "Operation" signal selects if inference or rule building sequence will be performed. If rule building is selected, the rules for each crisp state will be built according to the if-then rules of the inference operation. The generated crisp state rules will be sent to the RULE BANK where together with the the G matrix the fuzzy state rules are generated and stored. The recorded fuzzy state rules (R*Xn) are then accessed for the inference operation. Please refer to section 3.1.4 for detailed description on the inference module.

3.1.1 Reconfigurable HFB-FSM Module

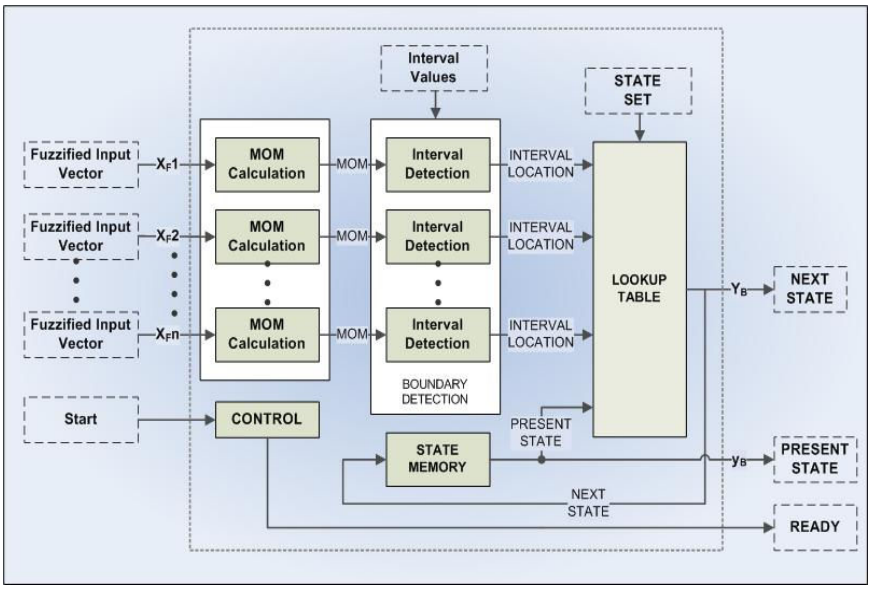

Figure 3.2 Reconfigurable HFB FSM Model Architecture

Figure 3.2 shows the reconfigurable HFB FSM architecture. The universal state space chosen for the system dictates the number of elements for each vector. On each clock cycle a member of each of the fuzzified input vector is placed onto its corresponding input. An external start signal is asserted to indicate the start of sampling of the vector inputs. The circuit counts the received elements of the fuzzy input vectors until the samples equal the expected value. This expected value is determined when the universal set is defined in the specification stage of the HFB-FSM. During the sampling of the elements of the fuzzy input vectors, the Mean of Maxima (MOM) operation is also executing. This scheme makes the MOM value of each of the fuzzy input vectors immediately available after the completion of the sampling of the elements. The MOM value is then passed into the Interval Detection logic where the MOM value's location (Interval Location) is determined from the pre-defined intervals. The Interval Location

together with the present state is compared to the lookup table to determine the next state of the automata.

3.1.2 Mean of Maxima Calculation

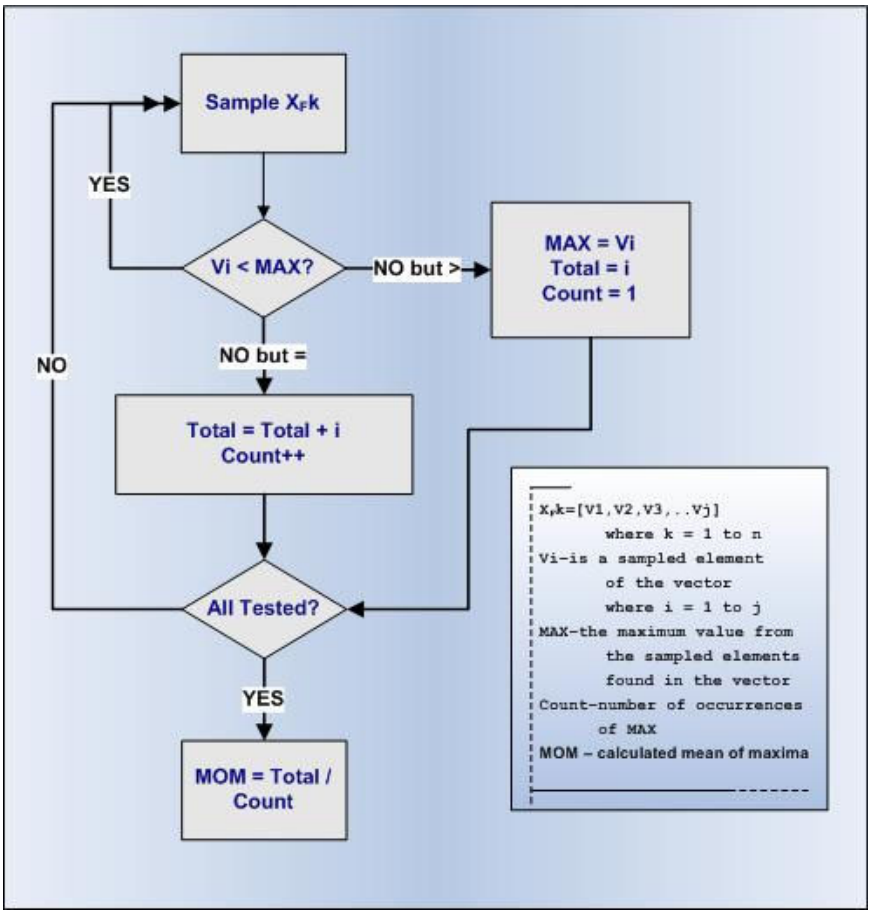

Figure 3.3 Mean of Maxima Calculation Algorithm

The calculations of the MOM is done incrementally while the fuzzy inputs are sampled. The algorithm is shown in Figure 3.3. The number of samples (j) is equal to the number of points in the universal set. The algorithm simply tests each sample (Vi) for a maximum value and records this value (assigned to MAX). Each time the maximum value is received, at a particular fuzzy input, the counter is incremented (Count). Also the corresponding sample count is added to a running sum (Total). When a new maximum

value is sampled, the new value will be recorded as the new maximum number and the running sum will be reset back to the corresponding sample count (Total = i) of the new value at the same time resetting the counter back to 1 (Count = 1).

For example if the current maximum value is 0.7 and the new value of 0.8 is detected at sample number 10 of the universal set space of 25 elements, the new value will replace the old one and the running sum will be set to 10. If 0.8 is found again at sample number 20 the running sum will be updated to 10 + 20 = 30. The final value of the counter that counts the number of occurrence of the maximum value will be used as the divisor for the running sum of the added sample counts. For the example of maximum value 0.8 the divisor will be 2 since it was detected only at sample numbers 10 and 20. Therefore, the computed MOM value will be 15. Since each fuzzy input is assumed to be available every clock, the clock signal should be generated in phase with the available data samples.

3.1.3 Interval Detection Module

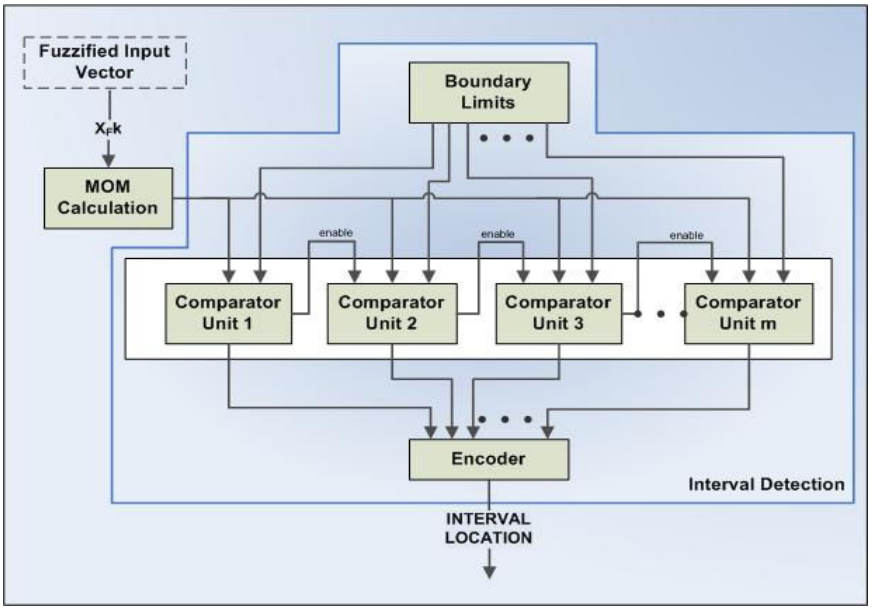

Figure 3.4 Interval Location Detection Logic

The interval detection module is implemented using cascaded comparators (see Figure 3.4). Each comparator compares its maximum limit with the MOM value. The next comparator is enabled when the MOM value is equal, or above its maximum. A value below the maximum limit does not enable the next comparator. The enable signals are then encoded to determine the sub-interval where the MOM value falls. For example, if there are five comparators and the concatenated enable signals yields a value of "11000" in binary, the encoded Interval Location value will be 2. The range of values of the interval location is from 0 to the number of boundaries – 1. The number of boundaries determines the number of comparators implemented to carry out the interval detection.

The cascaded comparators compares the MOM value with the pre-determined boundary limits. These limits are dependent upon the choice of the state transition conditions outlined in Chapter 2. For example, if the boundary limits are modified in such a way that the widths of the sub-intervals are widened or narrowed, the state transition can be modified. Figure 3.5 below shows new interval locations of MOM Value1 by modifying the boundaries.

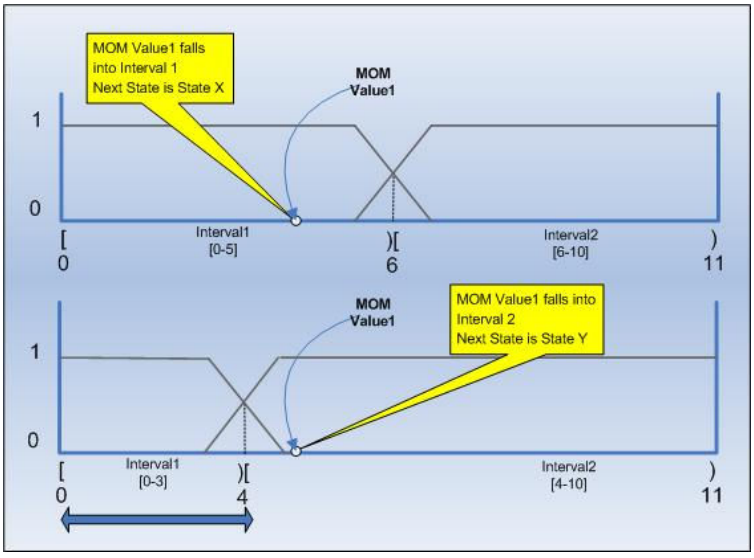

Figure 3.5 Modifying the Intervals to Change State Transition Conditions

To illustrate the modification, let us consider the boundary limits of 0-5 and 6-10, respectively, for a universal space with two Boolean sub-intervals. Let us further assume that a state transition from the present state to state x is performed if the MOM value of a fuzzy input falls in sub-interval 0-5, and the MOM evaluation produces a value of 4. However, we can effectively change the behavior of the system by changing the boundaries of the first Boolean sub-interval to 0-3. Hence, the state transition from the present state to state x will be modified to a transition to, say, state y.

3.1.4 Lookup Table Module

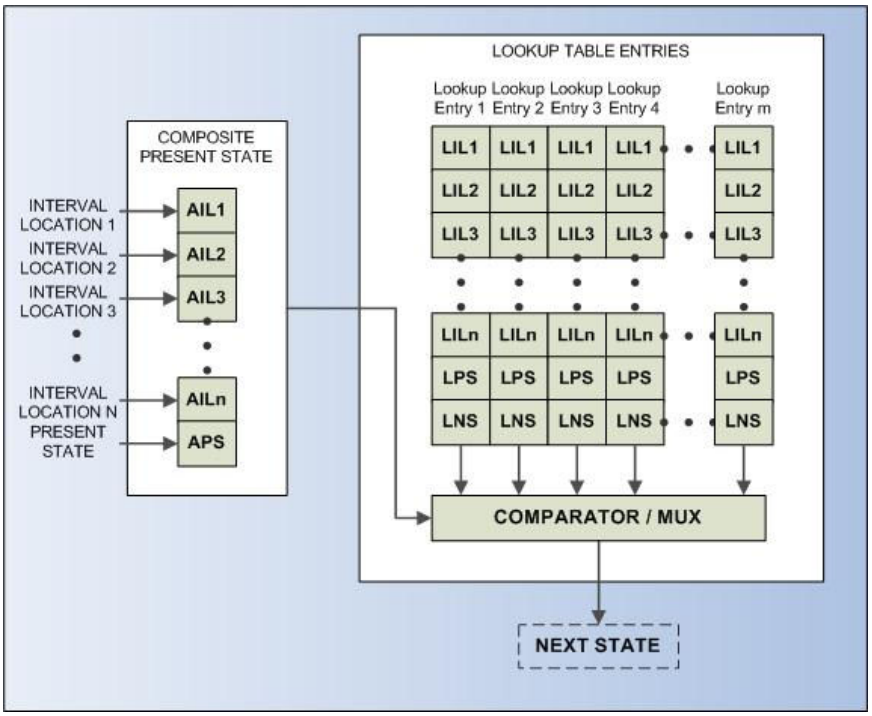

Figure 3.6 Lookup Table Implemention [1]

The calculations of the MOM values for each fuzzy input, fuzzy output and of the interval detections are performed in parallel, as shown in Figure 3.6. The resulting interval locations are concatenated along with the present state to form the composite input vector to the system. The composite input vector is represented as follows:

[AIL1:AIL2:...:AILn:PS] where AILi (i = 1, 2,...,n) stands for the ith actual interval location and the PS stands for the present state of the HFB FSM. The new composite input vector is compared with all the entries of the lookup-table. The lookup-table is a conscise representation of the behavior of the HFB FSM with respect to the defined state transitions through the dominant crisp states. The look-up table entries are given in the following vector format: [LIL1:LIL2:...:LILn:PS:NS] where LILi stands for the ith look-up interval location and the PS and NS stand for the present and next states, respectively. If the composite input vector matches an entry in the look-up table then the designated next state becomes the present state.

Theoretically, the lookup table can be extended infinitely. As more and more inputs and outputs are included into the system the list (or the n value) will simply grow. This growth of variables creates the problem of completely specifying the system. To put it plainly, more variables means more states. Therefore, it is important to identify the optimal partitioning of the states in the system state set that will be active during the operation. These states, together with the conditions for transitions in the inputs and outputs, will be the activated during reconfiguration of the automata. This is achieved through the programming of the lookup table.

3.1.5 Composition Module

The Composition Module is built from the basic inference and model building algorithm designed earlier in previous work [7] also described in Section 2.4. Taking advantage of the repetitive nature of the algorithm the design implemented loops to make the code compact but at the cost of longer combinational delays. The author decided on this implementation since industrial controllers are usually operating in the kHz range upto lower MHz range. Current technology of reconfigurable devices such as Xilinx' FPGA modules can operate upto 100 MHz which gives a reasonable operating frequency . This gives the system designer more room to work on tuning the module's clock and the system clock.

Unlike the previous work in [7], the implementation of the model building and inference module is separate. It does not take advantage of the fact that the minimum operation and maximum operation of the algorithm can be shared by both the inference and model building processes. The reason behind this decision is that the model building timing sequence is different than that of the inference timing sequence. Table 3.1 below illustrates this point. It shows the clock cycle performance of model building and inference processes. In the model building the rules needed to build the aggregated rules are fed into the module one at a time. This takes Sx7x7x5 clocks since it depends on the number of rules per state and the number of elements in the universal space. The S factor in the equation determines the number of times rule building is repeated to establish the rules for each state. The 1 + K portion is the constant overhead for any operation and another clock cycle to convert the aggregated crisp state rules to fuzzy state rules using the G-matrix to factor in the state membership. On the other hand, the inference operation takes only 7 + K clock cycles since the rules are already in place and ready for use. Only the fuzzy inputs are fed into the module one value per clock cycle. Since there are 7 elements it will take only this few cycles compared to the model building operation. Sharing the min and max circuit will result in a longer inference operation since the equation becomes (7x7)+ K.

Table 3.1 Inference and Model Building Operation Performance Summary.

Type of Operation	**Number of Rules per State**	**Size of 1 Fuzzy State Rule**	**Number of Elements in the Universal Set**	**Number Clock Cycles Needed**
Inference	5	7x7	7	7+K
Model Building	5	7x7	7	(Sx7x7x5)+1+K

Where: K is the constant overhead cycles when performing the operation
S is the number of states

The model building implementation (for N number of rules of a SISO configuration) is based on the algorithm [7] shown in C style coding below :

```
for (i = 1; i <= N; i++) {                                    (3.2)
```

for (j = 1; j <= P; j++) {

$temp_{ij}(u,w) = \min(X_i(u_j),Z_i(w_1)),$

$\min(X_i(u_j),Z_i(w_2)),...,$

$...,\min(X_i(u_j),Z_i(w_P));$

$rule_{ij}(u,w) = \max(temp_{ij}(u_j,w_1), rule_{(i-1)j}(u_j,w_1)),$

$\max(temp_{ij}(u_j,w_2), rule_{(i-1)j}(u_j,w_2)),...,$

$...,\max(temp_{ij}(u_j,w_P), rule_{(i-1)j}(u_j,w_P));$

}

}

Where $X_i(u_j)$ and $Z_i(w_j)$ are elements of the fuzzy inputs and fuzzy output sampled to create the inference rule. i = 1 to N; j = 1 to P;

The intermediate result ($temp_{ij}$) and the resulting rule is a vector of P elements. The aggregated rule is a matrix of size P x P initially set to zero at the start of rule building. From the algorithm the resulting $rule_{ij}$ is computed row by row. The hardware implementation of the algorithm uses only one port for the X_i and one port for the Z_i inputs. To effectively compute the rule, $temp_{ij}(u,w)$ vector is incrementally built one clock at a time (1 to P clock cycles). $Z_i(w_j)$ vector is iterated to each $X_i(u_j)$ inputs.

for (i = 1; i <= N; i++) { (3.3)

for (j = 1; j <= P; j++) {

$tempX = X_i(u_j);$

for (h = 1; h <= P; h++) {

$temp_{ijh}(u,w) = \min(tempX,Z_i(w_h);$

$rule_{ijh}(u,w) = \max(temp_{ijh}(u_j,w_h), rule_{(i-1)jh}(u_j,w_h));$

}

}

}

Where $temp_{ijh}(u,w)$ and $rule_{ijh}(u,w)$ are the elements of ith vector of the temporary variable and rule matrix. Equations 3.2 and 3.3 essentially yields the same rule. The only difference is that in Equation 3.2, the elements of the row of the rule is constructed parallel while in Equation 3.3 contructs the rule serially.

Rule building is performed for every crisp state. For example if there are 3 states and 2 inputs (MISO configuration), the total rules built is 3 x 2 = 6 rules. Each state will have 2 rules (one rule for each input in each state).

After the crisp state rule is constructed, the rule is transformed into fuzzy state rule (R*) by applying the G-Matrix. Equation 2.10 is used to perform this transformation. Since all the crisp state rules and the G-Matrix are available the transformation of the crisp state rules to fuzzy state rules is done in one clock cycle. The number of fuzzy state rules contructed is equal to the number of crisp state rules.

The inference algorithm (Equation 3.4) is similar to the rule building algorithm described above [7].

for (j = 1; j <= P; j++) { (3.4)

$t_j(u,w) = \min(X(u_j), R^*_j(u_j, w_1)),$

$\min(X(u_j), R^*_j(u_j, w_2)),...,$

$...,\min(X(u_j), R^*_j(u_j, w_P));$

$Z(w) = \max(t_j(u_j,w_1), Z(w_1)),$

$\max(t_j(u_j,w_2), Z(w_2)),...,$

$...,\max(t_j(u_j,w_P), Z(w_P));$

}

Where X (u_j) and Z (w_j) are elements of the fuzzy inputs and fuzzy output vectors respectively, j = 1 to P.

Z(w) is built incrementally every time X (u_i) is sampled. After the pth cycle, the inferred conclusion is ready at the output.

To implement the MISO configuration of the design for the composition module a SISO module is first constructed. The SISO structure is shown in Figure 3.7. This module consists of the model building, rule memory and the inference sub-modules.

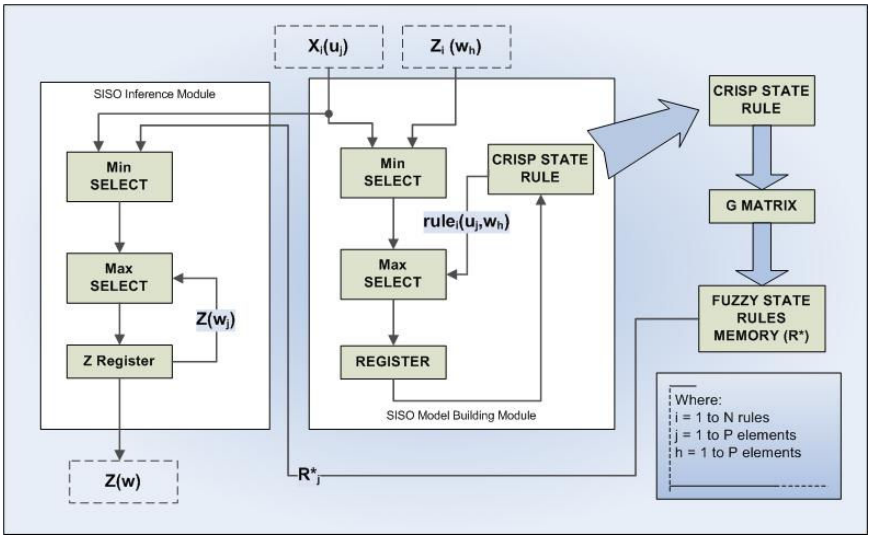

Figure 3.7 Basic SISO Module Structure

To implement the MISO Model Building components the Basic SISO Module is replicated depending on the number of inputs (See Figure 3.8). This implementation is advantageous in terms of verification since testing only one instance will verify all the rest. The operation of the MISO system is working in parallel. All the fuzzy inputs are sampled at the same time with the fuzzy output of each SISO component generated at the same time. Doing the minimum operation on each fuzzy output determines the final inferred result at the output.

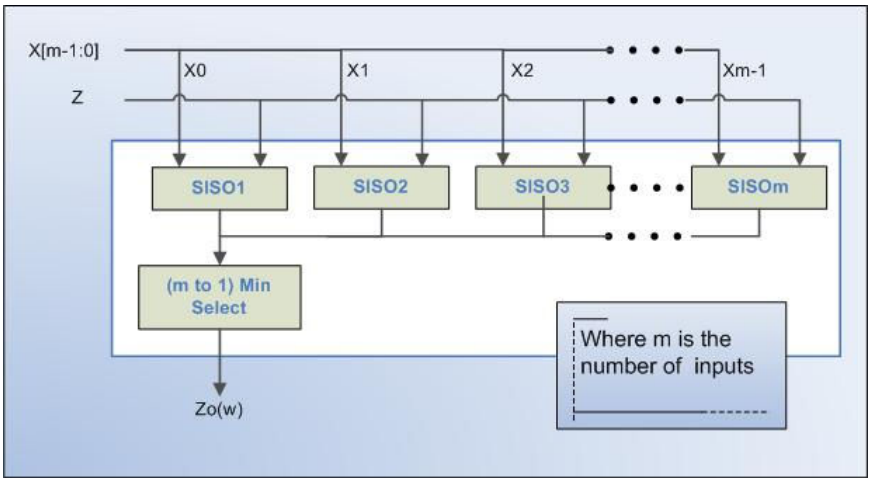

Figure 3.8 Composition Implementation

3.2 Reconfigurable Components

The main objective of the reconfiguration is to provide the system the ability to adapt to the varying clusters of states along the goal path and to model the plant. The circuit carries out the comparison of the present composite input vector with the entries of a reconfigurable look-up table. The proposed architecture introduces parameterized components into the design that add versatility to the implementation. The architecture can be implemented as a module in the system that performs the state transition decision of the HFB FSM.

Table 3.2 Parameterized Components

Component	Description
Number of Fuzzy Inputs	This defines the number of fuzzy inputs of the system. This value also determines the number of parallel MOM and Interval Detection circuits.
Resolution	The resolution of the degree of membership is resizable. This determines the number of bits needed to represent the degree of membership.
Granularity	This resizable property depends upon the number of elements in the universal set.
Boundary Count	This is the number of Boolean sub-intervals. It can be reset from problem to problem.
Boundary Limits	This is the right-most element of each Boolean sub-interval. The total number of limits is equal to the Boundary Count.
Number of Fuzzy States	The number of fuzzy states in the particular state cluster to be implemented.

The parameterized components of the proposed architecture are summarized in Table 3.2. A typical system has multiple inputs. The number of fuzzy inputs will differ from system to system. The width of the discrete representation of the degree of membership of each element of the universe of discourse determines the resolution of the fuzzified data. If the degree of membership is given by 4-bits then the resolution is 1/16. On the other hand, the number of elements in the universe of discourse determines the granularity of the fuzzy input. This number also represent the constraint on the number of possible Boolean sub-intervals.

3.3 Interfacing

The author assumes that the design will be interfaced to a system using the following interface signals.

Table 3.3 Interface Signal Definition

Signal Name	Signal Description
INPUTS	
clk	This is the clock signal of the module.
reset_l	This is a low asserted signal to reset the registers and rule memories of the module.
fzyin	This is the fuzzy input bus ($X_F k$). This is actually a two dimensional signal. Each component of the bus is one fuzzy input. The fuzzy values must be sent every clock cycle when "enable" signal is asserted.
zbusin	This is the Z bus input. Similar to fzyin, this is also a two dimensional signal. This signal is dedicated to rule building.
op	This signal defines the operation to be executed. This signal, together with the "enable" signal, informs the module if inference or model building will be performed. 0. Model Building Operation 1. Inference Operation
convert	This pulse is given by the host to the module to do conversion of the crisp state rules to fuzzy state rules.
enable	This enable signal starts the operation specified by "op". The host must maintain this signal asserted until all the fuzzy inputs are sent to the module.
momstart	This pulse signal indicates the start of MOM calculation for state transition determination.
fstate	This input is used in rule building to identify the state of the current rule being created.
OUTPUTS	
fzyout	This is the parallel fuzzy output signal (Z_F). After the inference operation, the values seen at this port is the result of the inference.
newstrdy	This asserted high output signal indicates that the next state decision is ready at the "fzy_nxt_state" port.
fzy_nxt_state	This output signal bus (Y_B) gives the chosen next state value for the current set of observed inputs (uses one-hot state implementation scheme). The width of this signal depends on the number of specified states.

3.3.1 Model Building Timing Diagram

A timing diagram of the model building sequence is shown in Figure 3.9. This particular example shows that there are 3 states for a SISO configuration. The external controller must assert the “enable” signal with the appropriate level of the “op” signal (op = ‘1’) for model building. The rules are driven into the “fzyin” and “zbusin” inputs every clock cycle. Three sets of rules are created (one for each state) from cycle 2 to cycle 15. Notice that the “fstate” input signal must be driven to indicate which of the state rules is being currently constructed. The fuzzy state rules are created from the aggregated crisp state rules by asserting the “convert” signal at cycle 18. Crisp state rules are ready for conversion as early as cycle 16. After the conversion of the crisp state rules into the fuzzy state rules, the design is ready to perform inference with next state detection.

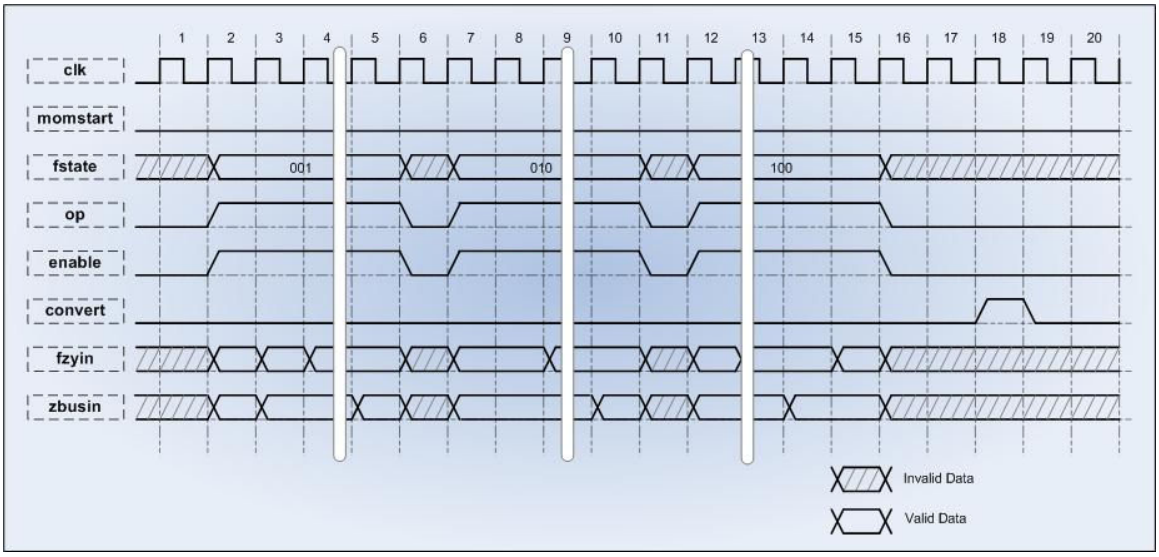

Figure 3.9 Model Building Timing

3.3.2 Inference and State Decision Timing Diagram

In the inference and state decision mode the “fstate” and “zbusin” signals are don’t care signals. Figure 3.10 shows an example of the timing of the mode. The “momstart” signal is asserted for one clock cycle to start the operation. Similar to the model building sequence, the “enable” signal must be asserted while valid data in the “fzyin” signal is available. The “fzyout” output data is available one clock cycle after the last “fzyin” data has been sampled. New state decision is available three clock cycles after the last fzyin data has been sampled. The “newstrdy” signal is asserted to indicate that the new state is ready for sampling by the host.

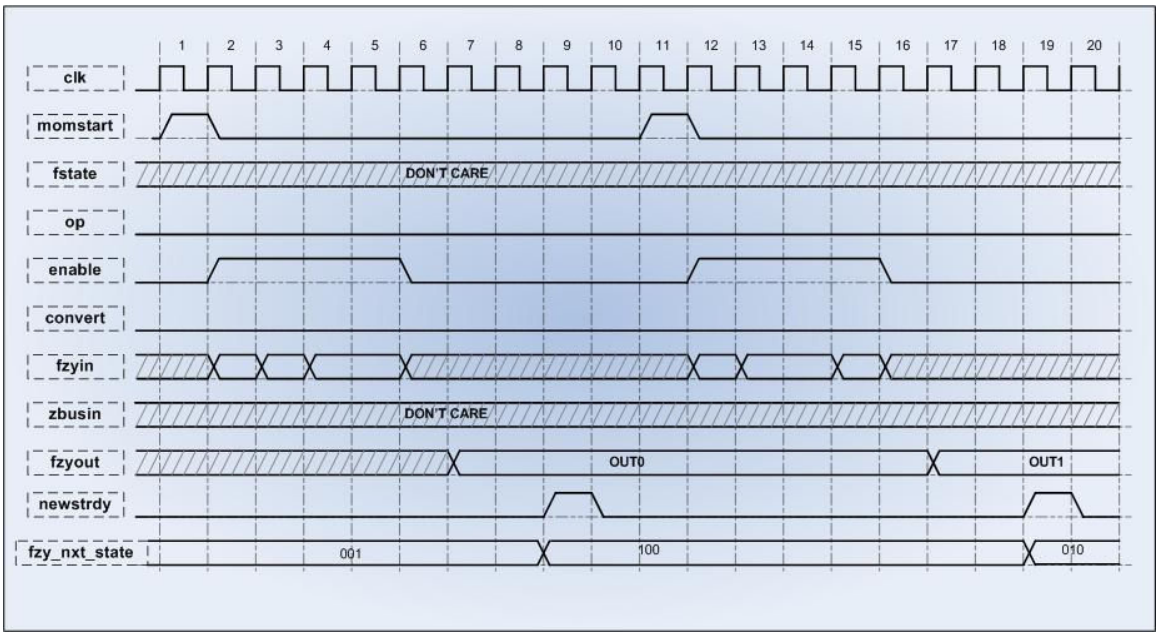

Figure 3.10 Inference and State DecisionTiming

3.4 System Integration

The reconfigurable architecture is intended to be implemented in systems that utilizes the HFB FSM functionality such as the ontological or supervisory controllers. A host controller will reconfigures the hardware as needed. Although the actual design of the host system is not the focus of this research a description on the possible implementation and structure of the system is described in the following paragraphs.

The design proposes simple protocol requirements mentioned in Sections 3.3.6 and 3.3.7. The host interface must provide the signals necessary to do rule building and inference. In the inference mode (with next state decision), the "newstrdy" signal must be monitored to detect when the next state information is ready. The inferred output can be sampled at the same time as the next state information.

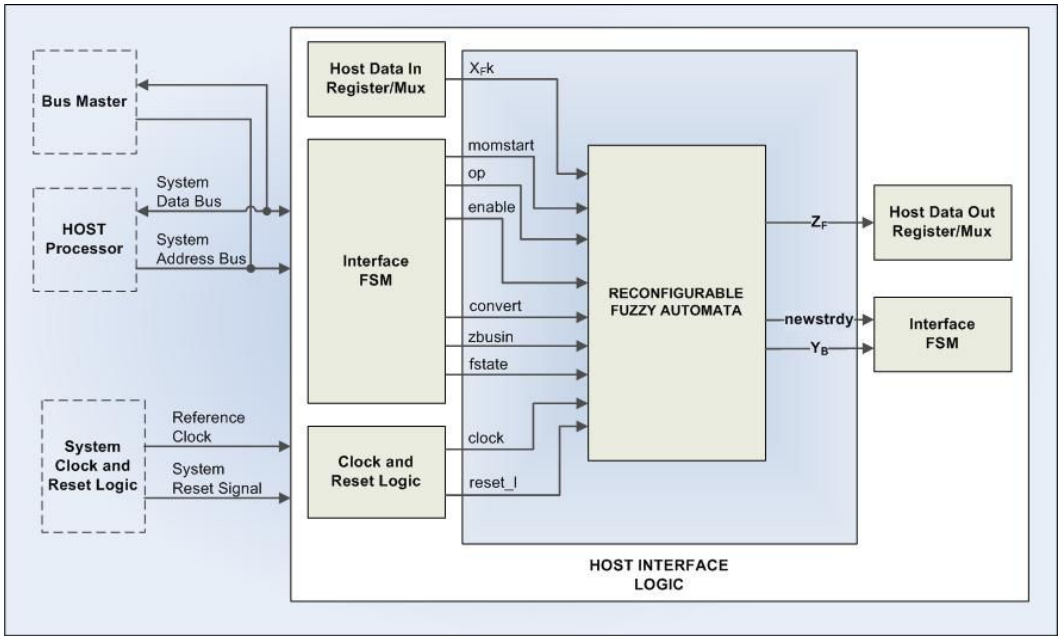

Figure 3.11 Sample System Implementation

The host interface circuitry must also be reconfigurable to take advantage of the reconfigurable feature of the design. Figure 3.11 shows an example system configuration

where the Reconfigurable Fuzzy Automata design is implemented. The Host Interface Logic (HIL) must be reconfigured together with the design. The reconfiguration can be performed by the Host Processor where the Ontological Controller (host) program is executing. Reconfiguration will be performed accordingly depending observed values of the inputs and state of the system.

The design of the HIL must meet the interface requirements of the design. The fuzzy inputs ($X_F k$) will be multiplexed and sent to the design in parallel. The data registers that receives the fuzzy inputs can be assigned addresses (I/O Mapped) and the data can be written to the HIL using the System Processor or using other system bus master processing the inputs. A FSM in the HIL can initiate and control the operation of rule building and inference. Sampling of the fuzzy outputs (Z_F) and next state information (Y_B) is performed by the FSM and sent to the host program for use.

The system bus can be any generic bus that support the access to the HIL. From the system configuration we can see that the system clock frequency will be much faster than the clock frequency used in the Reconfigurable Fuzzy Automata.

Reconfigurable devices such as the Xilinx's Virtex Pro FPGA can be used to implement such system where an internal CPU and a system bus is provided. The FPGA also supports partial reconfiguration on the FPGA which is a very attracted feature that will take advantage on the reconfigurable feature of the design.

CHAPTER 4

DESIGN MODELING AND VERIFICATION

4.1 VHDL Design

To implement the design in a reconfigurable manner, the use of parameterized components is widespread in the implementation. There are two ways to implement parameterized design in VHDL. First is through the use of generics and second is through the use of new data types with constants. From the implementation point of view, the use of generics provides the parameterization of the design that is needed to make it reconfigurable but the author chose the use of new data types with constants since this will make coding straightforward. Using new data types adds coding versatility otherwise not possible with classic coding. This section describes the implementation of the code in detail. It highlights the components that implements the reconfigurable feature of the design.

4.1.1 Parameters Package

The heart of the code is the package named “parameters.vhd”. All the declared constants and parameters specified to implement the design into hardware is found in this package file. Values of the parameterized components listed in Table 3.1 are specified in this file. Starting from the definition of constants that define parameters such as signal width, number of intervals in the universal set and number of states, the new data types are declared. These data types are specifically created to facilitate a reconfigurable architecture. Synthesizing the design with the specified parameters based on the specification of the system creates the hardware implementation. Below is a portion of the Parameters Package code that implements the width of the fuzzy input and the aggregated signals that are created based on this type.

```
--This is the fuzzy input width
 constant FUZZY_WIDTH  : integer := 4;

-- This is the type that defines a fuzzy value
type FUZZY_VECTOR is
 array((FUZZY_WIDTH-1) downto 0) of bit;

-- FUZZY VECTOR in bus form
type FUZZY_VECTOR_BUS is
 array((NO_FUZZY_INPUTS-1) downto 0) of FUZZY_VECTOR;

-- data type for rule memory
-- this creates a memory array of FUZZY_VECTOR of size FUZZY_VEC_WIDTH
type rule_memory is array((
            (FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) downto 0)
             of FUZZY_VECTOR;

-- this creates the array of rule_memory depending
-- on the number of states
type rule_mem_array is  array(NO_FUZZY_STATES-1 downto 0)
             of rule_memory;
```

As shown later, this implementation facilitates signal manipulation especially in generating signals and modules to implement multiple instances.

The Parameters Package also includes the initialization of the contents of the lookup table that specifies the conditions for state trasition and the contents of the G matrix used to denote the degree of state membership functions.

4.1.2 HFB-FSM Top Module

This module instantiates the two main components (mibtop.vhd and parallel_infsiso.vhd) of the design. The mibtop component is the instance of the portion of the HFB-FSM that implements the state transition determination module that advises the ontological controller on the state transition of the system. The parallel_infsiso module on the other hand is the instance of the inference engine including the storage for the fuzzy state rules described in Section 2.4.

4.1.3 Mibtop Module

The mibtop module in turn implements the lookup table, the MOM calculation, interval detection and lookup table. Since the lookup table size, number of intervals is variable from one system to another, the author used the generate statement to create the instances of the module. The table size and interval limits are taken from the Parameter Package. Below is the code that implements the instances of the MOM calculation and Interval Detection pairs that computes for the interval location of the fuzzy inputs.

```
-- this is the generation of the MOM-sisotop pairs to generate the
-- interval location of each of the fuzzy input vectors
momsiso: for i in 0 to (NO_FUZZY_INPUTS-1) generate
   mom: meanofmax port map (fzy_inpts(i), momstart, clk,
                            reset_l, done(i), fzymombus(i));
   stop: sisobtop port map (fzymombus(i),fzyintvl(i));
end generate momsiso;
```

NO_FUZZY_INPUTS is a constant defined in the Parameters Package that specifies the number of fuzzy inputs in the system..

4.1.4 Parallel_infsiso Module

This module was also implemented using the basic inference module for SISO design. To implement the MISO version the SISO components are instanced in parallel using again the generate statement. The tricky portion of the design is in the implementation of the SISO portion. The design included the instance of the rule memory component. This gave freedom in coding since manipulations for easier rule access in performing the min and max operations is now possible. Another advantage is that execution of some repetitive functions, such as the min and max operations, can be performed in parallel in one clock cycle. The drawback of this choice is that the design tends to grow significantly when more and more rules are added into the design. This may pose a problem in the device implementation later. Below is the implementation of a min operation.

```
tstate_rule_gen: process(memrule, tmemrule)
begin
   for i in 0 to (NO_FUZZY_STATES-1) loop
      for j in 0 to (NO_FUZZY_STATES-1) loop
         for k in 0 to ((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) loop
            if (memrule(j)(k) > g_matrix((i*NO_FUZZY_STATES)+j)) then
               tmemrule(i)(j)(k) <= g_matrix((i*NO_FUZZY_STATES)+j);
            else
               tmemrule(i)(j)(k) <= memrule(j)(k);
            end if;
         end loop;
      end loop;
   end loop;
end process;
```

Notice that the implementation is a combinational logic that generates the fuzzy state rules (tmemrule) from the crisp state rules (memrule) and the G matrix (g_matrix). The operation is actually taking the minimum value between an element in the crisp state rule

matrix and a constant from the G matrix. The operation is repeated in every state since each crisp state has its rule matrix.

4.2 Verification Strategy

Since the architecture is composed of two major components, mibtop and parallel_infsiso modules, the verification was also divided into two main phases. The first phase involved the verification of the mibtop module. This is the module that implements the determination of the next state based on the observed fuzzy inputs and the present state. The second phase involved the verification of the parallel_infsiso module which is the inference and model building implementation.

To help in the testing, a software model that implements part of the HFB FSM was used to counter check the hardware verification results. Both the inference simulations and the state transition simulations were run in both hardware and software version of the design. The hardware verification was performed using "do" files for simulations on Mentor Graphics' ModelSim Simulator.

4.3 Hardware Model with Software Model Verification

4.3.1 Eye-Hand Coordination Example

A software model that implements part of the HFB FSM is being used to develop an Intelligent Decision Support System (IDSS) for Eye-Hand Coordination Assessment. The initial results of the research were reported in [6]. The goal is to develop an automated assessment and training procedure for children with eye-hand coordination problems. This software model was used to verify the operation of the hardware implementation. By specifying the same parameters that are being used by the software model it is possible to confirm the hardware implementation works properly. The state transitions made by the hardware implementation are compared with the ones exhibited by the software model.

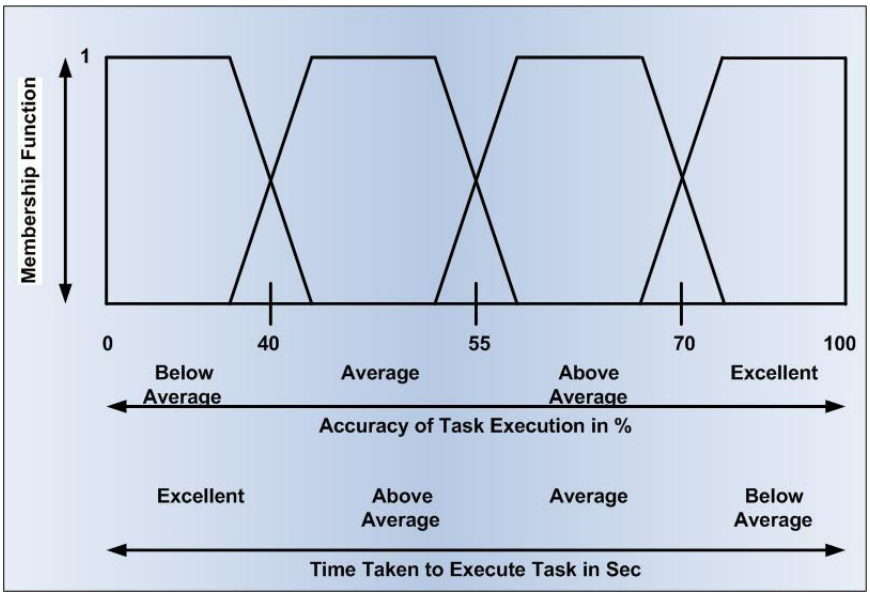

Figure 4.1 Normalized Universal Sets for Time and Accuracy Inputs [1]

The universal set used in both the software model and the reconfigurable hardware implementation is shown in Figure 4.1. The Boolean sub-intervals used for the fuzzy inputs Accuracy (given in percentile) and Time (given in seconds) are normalized to just one uniform set to simplify the hardware implementation. Accuracy is defined with four sub-intervals ranging from Below Average to Excellent while the Time is also defined with four sub-intervals ranging from Excellent to Below Average. The upper bounds for the sub-intervals are also shown in Figure 4.1. The defuzzified values for the inputs Time and Accuracy are shown in Figures 4.2 and 4.3, respectively. The state transition graph to be implemented by the HFB FSM is shown in Figure 4.4. This state transition graph has been developed for children of age 5. It is based on experimental knowledge provided by Occupational Therapy experts.

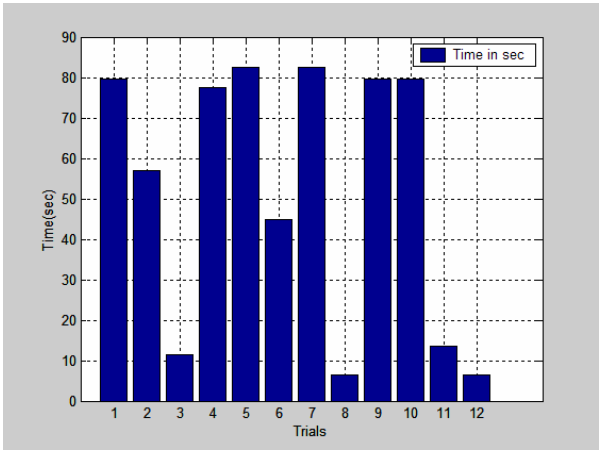

Figure 4.2 Defuzzified Time Input in Seconds [1]

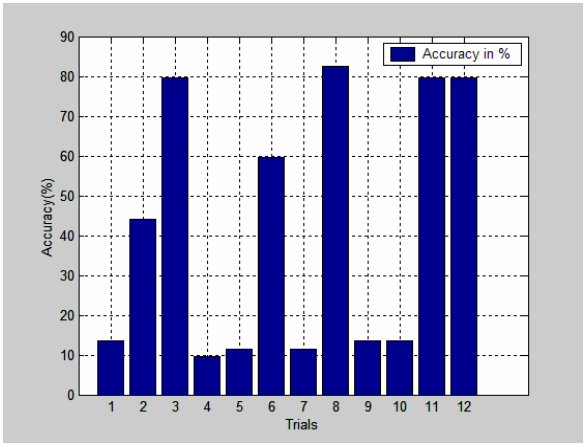

Figure 4.3 Defuzzified Accuracy Input in Percentile [1]

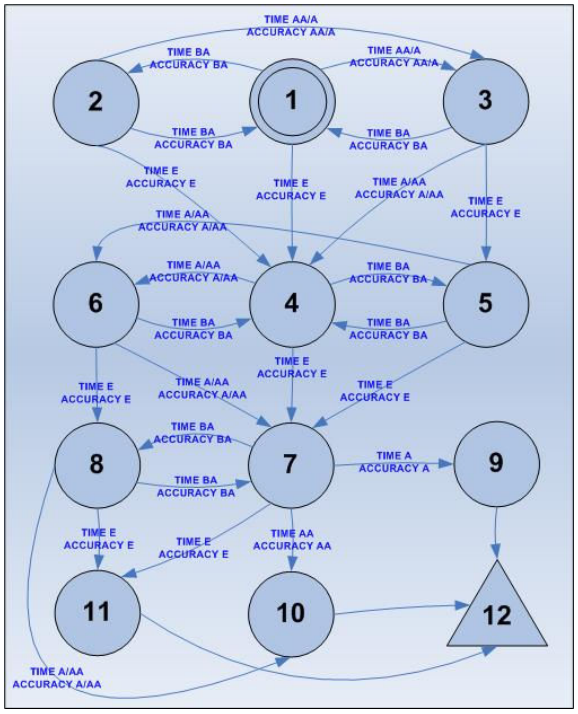

Figure 4.4 State Transition Graph [1]

Supporting the whole research project requires a reconfigurable software/hardware model, which can adapt the state transition graph to the age group that is being assessed. The state transition graph depicts a path to improved hand-eye coordination. Starting at state1 and reaching state12 without any repetitions indicates a significant improvement in the subject's eye-hand coordination skills, the state transition graph is traversed such that the next state depends upon the present state and the accuracy and time inputs received in a trial. Each pair of inputs represents one trial.

The recorded state transitions in the hardware simulation were tabulated into a file and compared with the results of the software model. It has been found that the results of both the software simulations and hardware simulations are identical as shown in Figure 4.5.

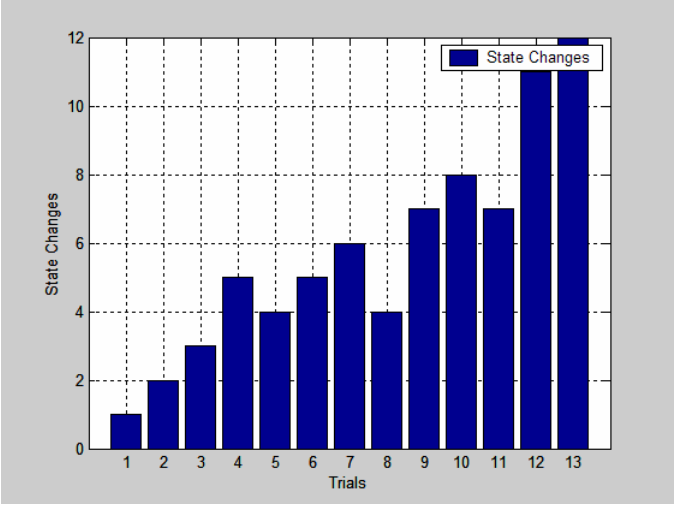

Figure 4.5 State Visited During Trials [1]

The x-axis shows the trial number and the y-axis shows the states visited by the HFB FSM. Initially, the HFB FSM is in state 1. Using the Time and Accuracy inputs shown in Figures 4.2 and 4.3 for trial 1, the state of the HFB FSM did not change. At trial 2 and using new values for the Time and Accuracy inputs, the state changed from state 1 to state

3. The simulation reached state 12 with 13 trials using the Time and Accuracy inputs shown in Figures 4.2 and 4.3, respectively.

The look-up table implemented in both the software model and the hardware implementation uses the AND compression of the states of the two fuzzy inputs to trigger a state change. This is one limitation of the current hardware implementation of the fuzzy automaton. Another current limitation of the hardware is that the fuzzy inputs must be normalized to the same universal set.

The simulation was performed using the same set of inputs, lookup table and state transition graph for both the reconfigurable hardware design and the software model. No special test bench was created to check that the timing requirements of the hardware design would meet the system requirements of the IDSS. The inputs were converted to the format that the hardware expects. Similarly, the output of the hardware design was converted back to the same format of the output of the software model to perform the comparison. For example, the 0.5 input was converted to 0x8 in hex since the hardware input width used was 4-bits.

4.3.2 Container-Crane Simulation Example

Another example developed to test the hardware design is patterned after the Container-Crane simulation example given by FuzzyTECH 5.5 [13]. The Container-Crane problem (see Figure 4.6) simulates the operation of transferring a container van from a ship into a railcar platform. Due to the heavy weight of the transferred container a great deal of feedback is experienced by the crane. This poses a big problem in controlling the transfer of the container when speed is the issue. Due to a large volume of containers that are being loaded and unloaded, finding the optimum speed that will not cause any catastrophic accident is essential.

The input variables chosen for the problem are the angle of swing and the distance. The output variable is the power applied to the crane. To simplify the problem, the angle of swing are assigned negative (left swing), zero (stationary) or positive value (right swing). The distance is divided into far, near and almost or close to the destination. The crane's power is also classified into medium positive, medium negative and zero representing movement of the crane to the right, left and no movement respectively. The system is then assumed to have three states defined by the distance of the container to its destination. States 1, 2 and 3 are assigned to far, near and close, respectively.

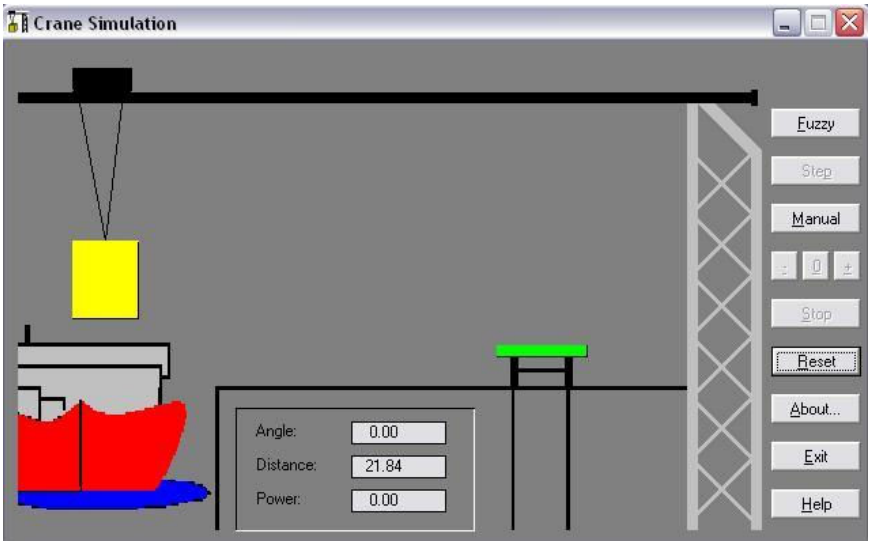

Figure 4.6 Container-Crane Problem [13]

Since all the three variables have three possible values it is easy to normalize the linguistic models of the three into one. The normalized universal set is shown below:

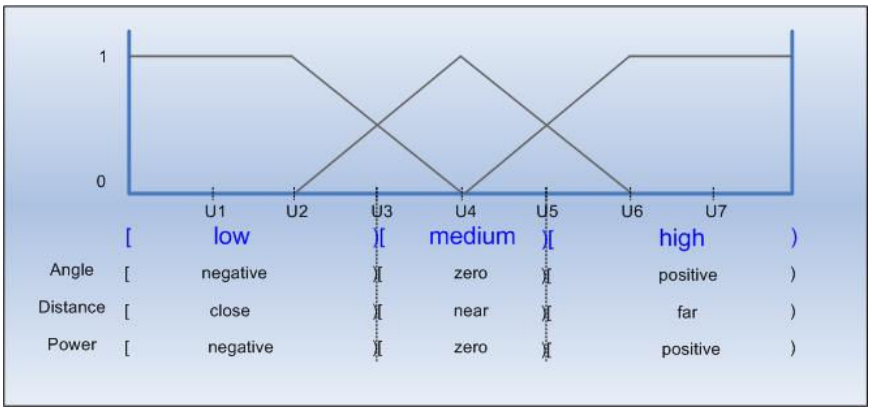

Figure 4.7 Normalized Universal Set for the Container-Crane Problem

Angle is in degrees, distance is in feet and power is in KiloWatt. To develop the linguistic model, let X be the angle, Y be the distance and Z be the power. The rule set are summarized in Table 3.4.

Table 4.1 Defined Liguistic Model for the Container-Crane Problem

Rules	Rule Description
For Crsip State 1 (far)	
Rule 1	If X is zero and Y is far then Z is **positive**.
Rule 2	If X is negative and Y is far then Z is **positive**.
Rule 3	If X is positive and Y is far then Z is **positive**.
For Crisp State 2 (near)	
Rule 1	If X is zero and Y is near then Z is **positive**.
Rule 2	If X is negative and Y is near then Z is **positive**.
Rule 3	If X is positive and Y is near then Z is **negative**.
Rule 4	If X is zero and Y is close then Z is **zero**.
Rule 5	If X is positive and Y is close then Z is **negative.**
Rule 6	If X is negative and Y is close then Z is **zero**.
For Crisp State 3 (close)	
Rule 1	If X is zero and Y is close then Z is **zero**.
Rule 2	If X is positive and Y is close then Z is **negative**.
Rule 3	If X is negative and Y is close then Z is **zero**.

In State 2, the rules to determine the power when transitioning from State 2 to State 3 are included. In State 1, rules to determine the power when transitioning from State 1 to State 2 are not included. This is done to illustrate the case of an undefined condition occuring in

the inputs. It will be shown later that the fuzzy outputs in this case are can still produce a desirable output due to the influence of the other defined rules.

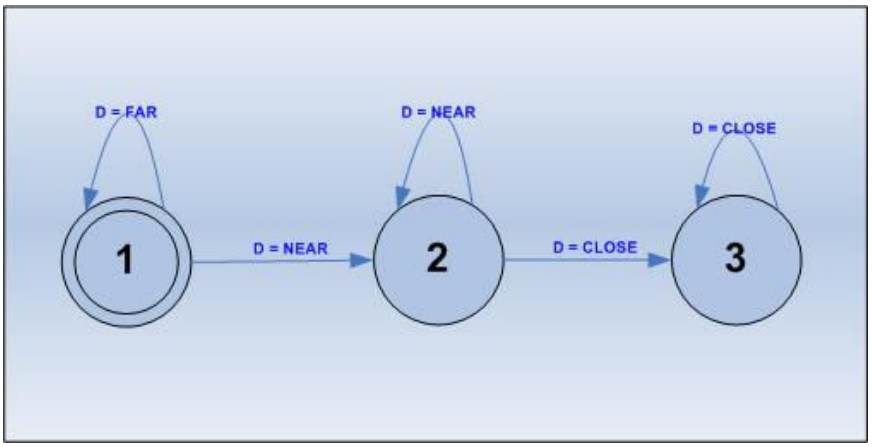

Figure 4.8 State Transition for the Container-Crane Problem

The state transition is illustrate in Figure 4.8. The state transition of the controller will be determined by the current distance of the container to its destination. State movement is only in one direction because of the nature of the problem. That is the reason why some of the fuzzy outputs in table 3.5 are undefined. For example, movement from State 3 to State 2, by design, should occur.

Table 4.2 Container-Crane State Transition Conditions and Fuzzy Outputs Summary

Present State	Next State	Fuzzy Input (Angle)	Fuzzy Input (Distance)	Fuzzy Output (Power)
1	1	Medium	High	High
1	1	Low	High	High
1	1	High	High	High
1	2	Low	Medium	Undefined
1	2	Medium	Medium	Undefined
1	2	High	Medium	Undefined
1	3	Low	Low	Undefined
1	3	Medium	Low	Undefined
1	3	High	Low	Undefined
2	2	Medium	Medium	High
2	2	Low	Medium	High

2	2	High	Medium	Low
2	1	Low	High	Undefined
2	1	Medium	High	Undefined
2	1	High	High	Undefined
2	3	Low	Low	Medium
2	3	Medium	Low	Medium
2	3	High	Low	Low
3	3	Medium	Low	Medium
3	3	Low	Low	Medium
3	3	High	Low	Low
3	2	Low	Medium	Undefined
3	2	Medium	Medium	Undefined
3	2	High	Medium	Undefined
3	1	Low	High	Undefined
3	1	Medium	High	Undefined
3	1	High	High	Undefined

Where: High = [0.0 0.0 0.0 0.0 0.5 1.0 1.0]
Medium = [0.0 0.0 0.5 1.0 0.5 0.0 0.0]
Low = [1.0 1.0 0.5 0.0 0.0 0.0 0.0]

Siumulation results are given in Table 3.6. Ten transitions were performed with the given Fuzzy Inputs and the fuzzy output values conformed with the rules established earlier. Output for state transition 1 to 2 was not defined in the rules. The corresponding fuzzy output shows that the power to be used was still calculated to be at positive power (high) or movement of the crane from left to right.

Table 4.3 Container-Crane Problem State Transition and Fuzzy Outputs Results

Present State	Next State	Fuzzy Input (Angle)	Fuzzy Input (Distance)	Fuzzy Output (Power)	Deffuzified Value
1	1	Medium	High	[0.0 0.0 0.0 0.0 0.5 1.0 1.0]	6.5
1	1	Low	High	[0.0 0.0 0.0 0.0 0.5 1.0 1.0]	6.5
1	1	High	High	[0.0 0.0 0.0 0.0 0.5 1.0 1.0]	6.5
1	2	Low	Medium	[0.0 0.0 0.0 0.0 0.5 0.5 0.5]	6
2	2	Medium	Medium	[0.0 0.0 0.5 0.5 0.5 1.0 1.0]	6.5
2	2	Low	Medium	[0.0 0.0 0.5 0.5 0.5 1.0 1.0]	6.5
2	2	High	Medium	[1.0 1.0 0.5 0.5 0.5 0.5 0.5]	1.5
2	3	Medium	Low	[0.5 0.5 0.5 1.0 0.5 0.5 0.5]	4
3	3	High	Low	[1.0 1.0 0.5 0.5 0.5 0.0 0.0]	1.5
3	3	Low	Low	[0.0 0.0 0.5 1.0 0.5 0.0 0.0]	4

The deffuzified values of the fuzzy output was obtained manually. The process is similar to the B-algorithm discussed earlier. From the calculated values, the power to the crane initially applied is positive medium (high) and continued until distance is near the destination. When the distance is closing at the destination, the power applied now varies from zero (medium) to negative (low) to adjust the position of the crane.

Again, this simulation is theoretical since the system requirements of the crane setup was not identified and verified to see that the reconfigurable hardware will meet those requirements.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This thesis presents an architecture design of a reconfigurable state transition algorithm for fuzzy automata, specifically the Hybrid Fuzzy-Boolean Finite State Machine (HFB FSM). The architecture presented establishes the framework of a virtual fuzzy automaton design. To realize the reconfigurable hardware implementation of the design features of the industry standard Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) were utilized. With parameterized components, a great degree of versatility with respect to the number of fuzzy inputs, as well as granularity and resolution of each fuzzy input is achieved. This represents a very attractive property to implement virtual fuzzy automata for supervisory controllers of complex systems.

For this work a previously developed software model was used; test vectors generated and comparison performed. The hardware implementation was verified since the simulation results of the hardware version matched those of the software model.

5.2 Future Work

Current design implementation can be improved in some areas. First is the implementation of the rule memory. Currently, the rule memory is integrated into the inference engine of the design. This scheme proved to be efficient in designing parallel operations but is limited to some degree in terms of operating clock frequency. As the rules increases in size, delay in the circuit also increases. An option in implementing the rule memory is to store it in a memory fashion (i.e. each element in the rule set is addressable). This scheme,

although it will require a state machine to control the addressing, should be acceptable due to improvement in speed and performance of FPGA technologies.

Second item that can be further researched is the systematic determination of the elements of the G matrix. Currently, the elements are chosen in an ad hoc fashion. The choice of value is currently dependent on the system and the experience of the designers or experts on the behavior of the system. These are usually based on the experiences of these resource persons. If the system behavior can be modeled in such a way that the G matrix elements can be derived systematically, it can be a very good tool to tune the operation of the system.

Next item is the selection of the next state. The current implementation uses a comparator to check for the current observed inputs and the entries in the lookup table. This implementation allows only a one-to-one matching to make a decision. Future implementation can include in the selection the use of do not cares and the OR compression to link the states of the inputs. This new feature will add more versatility to controllers with multiple inputs. In addition, different universal sets for each input will also be implemented.

Lastly, the current implementation of the universal set is normalized. This means that all the fuzzy inputs shares the same interval locations and number of intervals in the universal set. If each of the inputs can have its own unique values (i.e. no normalization), tuning of the state transitions can further be enhanced.

BIBLIOGRAPHY

[1] Grantner, J.L., Fodor, G.A., Tamayo, P.A., Gottipati, R. "Design of a Reconfigurable State Transition Algorithm for Fuzzy Automata", NAFIPS-'05. *Proceedings of the 2005 North American Fuzzy Information Processing Society Conference on* , 22- 25 June 2005

[2] Grantner, J.L., Fodor, G.A., "Fuzzy Automaton for Intelligent Hybrid Control Systems", FUZZ-IEEE'02. *Proceedings of the 2002 IEEE International Conference on* ,Volume: 2 , 12-17 May 2002 Pages:1027 – 1032

[3] Grantner, J.L., Fodor, G., Driankov, D., "Hybrid Fuzzy-Boolean Automata for Ontological Controllers"**,** Fuzzy Systems Proceedings, 1998. *IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on* , Volume: 1 , 4-9 May 1998 Pages:400 - 404

[4] Grantner, J.L., Patyra, M.J., "Reprogrammable Fuzzy Logic Finite State Machine Model", NAFIPS. 1996 *Biennial Conference of the North American Fuzzy Information Processing Society*, 19-22 June 1996 Pages:492 – 496

[5] Fodor G., "Ontologically Controlled Autonomous Systems: Principles, Operations and Architecture". Kluwer Academic Publishers, Boston/Dordrecht/London 1998

[6] Grantner J. L., Gottipati R., Pernalete N., Fodor G., Edwards S., "Intelligent Decision Support System for Eye-Hand Coordination Assessment". 2005 *FUZZ-IEEE Conference*, Reno, Nevada, May 22-25, 2005

[7] Grantner J. L., "Design of Event-Driven Real-Time Linguistic Models Based on Fuzzy Logic Finite State Machines for High-Speed Intelligent Fuzzy Logic Controllers". 1993 *Thesis for Degree of Technical Science*, Hungarian Academy of Sciences, Budapest , Hungary

[8] Grantner, J.L., Patyra, M.J., "VLSI Implementation of fuzzy logic finite state machines", IFSA '93 World Congress, Seoul, Korea, July 4-9, 1993, Proceedings Vol. II, pp. 781 – 784.

[9] Grantner, J.L., Fodor, G., Driankov, D., Patyra, Marek J., "Application of the Fuzzy State Fuzzy Output Finite State Machine to the Problem of Recovery from Violations of Ontological Assumptions"**,** paper to the ANNIE'96 Conference, St. Louis, MO, November 10-13, 1996

[10] Grantner, J.L., Fodor, G., "The Virtual Automaton Approach to the Problem of Global State Evaluation in Multitask Control Systems", Fuzzy Systems Proceedings, 1999. *IEEE World Congress on Computational Intelligence., The 1999 IEEE International Conference on* , Volume: 1 , 22-25 August 1999 Pages:283 – 288

[11] Klir, G. Yuan, B., "Fuzzy Sets and Fuzzy Logic – Theory and Applications", Upper Saddle River, New Jersey, Prentice-Hall Inc., 1995.

[12] Kandel, A., Langholz, G., "Fuzzy Control Systems", Boca Ranton, Florida, CRC Press Inc., 1994

[13] Inform Software Inc.., "FuzzyTECH User's Manual", Inform Software Corporation , 2001

[14] Bhasker, J., “A VHDL Primer”, Englewood Cliffs, New Jersey, Prentice-Hall Inc., 1995

Appendices

Appendix A – Implementation Hierarchy

==========

The structural hierarchy of the implementation is shown here. This is taken from the implementation of the SISO example described in Section 2.4.

==========

```
hfbfsm_top: hfbfsm_top(behavioral)
       inference: parallel_infsiso(structural)
              Generate parallel__0
                     par_inf: basic_infsiso(behavioral)
              Generate parallel__1
                     par_inf: basic_infsiso(behavioral)
              min: multin_min_sel(behavioral)
       mombfsm: mibtop(structural)
              Generate limits__0
              Generate limits__1
              Generate limits__2
              Generate limits__3
              Generate limits__4
              Generate limits__5
              Generate limits__6
              Generate limits__7
              Generate limits__8
              Generate limits__9
              Generate limits__10
              Generate limits__11
              Generate limits__12
              Generate limits__13
              Generate limits__14
              Generate limits__15
              Generate limits__16
              Generate limits__17
              Generate limits__18
              Generate limits__19
              Generate limits__20
              Generate limits__21
              Generate limits__22
              Generate limits__23
              Generate limits__24
              Generate limits__25
              Generate limits__26
              Generate momsiso__0
                     mom: meanofmax(behavioral)
                            divide: div_unit(rtl)
                     stop: sisobtop(structural)
                            bndrygen_ins:
bndrygen(behavioral)
                                   Generate limits__0
                                   Generate limits__1
                                   Generate limits__2
                            fzytobool_ins:
fzytobool(structural)
                                   Generate
comparators__0
                                          Generate
first_comp
                                                 comp1:
basiccomp(behavioral)
                                   Generate
comparators__1
                                          Generate
next_comp
                                                 comps:
basiccomp(behavioral)
                                   Generate
comparators__2
                                          Generate
next_comp
                                                 comps:
basiccomp(behavioral)
                            fzyinencode_ins:
fzyinencode(behavioral)
              Generate momsiso__1
                     mom: meanofmax(behavioral)
                            divide: div_unit(rtl)
                     stop: sisobtop(structural)
                            bndrygen_ins:
bndrygen(behavioral)
                                   Generate limits__0
                                   Generate limits__1
                                   Generate limits__2
```

```
                                fzytobool_ins:
fzytobool(structural)
                                        Generate
comparators__0
                                                Generate
first_comp
                                                        comp1:
basiccomp(behavioral)
                                        Generate
comparators__1
                                                Generate
next_comp
                                                        comps:
basiccomp(behavioral)
                                        Generate
comparators__2
                                                Generate
next_comp
                                                        comps:
basiccomp(behavioral)
                                fzyinencode_ins:
fzyinencode(behavioral)
                Generate prsnt_state__0
                Generate prsnt_state__1
                Generate lookup__0
                        lkup: lkup_cmp(behavioral)
                Generate lookup__1
                        lkup: lkup_cmp(behavioral)
                Generate lookup__2
                        lkup: lkup_cmp(behavioral)
                Generate lookup__3
                        lkup: lkup_cmp(behavioral)
                Generate lookup__4
                        lkup: lkup_cmp(behavioral)
                Generate lookup__5
                        lkup: lkup_cmp(behavioral)
                Generate lookup__6
                        lkup: lkup_cmp(behavioral)
                Generate lookup__7
                        lkup: lkup_cmp(behavioral)
                Generate lookup__8
                        lkup: lkup_cmp(behavioral)
                Generate lookup__9
                        lkup: lkup_cmp(behavioral)
                Generate lookup__10
                        lkup: lkup_cmp(behavioral)
                Generate lookup__11
                        lkup: lkup_cmp(behavioral)
                Generate lookup__12
                        lkup: lkup_cmp(behavioral)
                Generate lookup__13
                        lkup: lkup_cmp(behavioral)
                Generate lookup__14
                        lkup: lkup_cmp(behavioral)
                Generate lookup__15
                        lkup: lkup_cmp(behavioral)
                Generate lookup__16
                        lkup: lkup_cmp(behavioral)
                Generate lookup__17
                        lkup: lkup_cmp(behavioral)
                Generate lookup__18
                        lkup: lkup_cmp(behavioral)
                Generate lookup__19
                        lkup: lkup_cmp(behavioral)
                Generate lookup__20
                        lkup: lkup_cmp(behavioral)
                Generate lookup__21
                        lkup: lkup_cmp(behavioral)
                Generate lookup__22
                        lkup: lkup_cmp(behavioral)
                Generate lookup__23
                        lkup: lkup_cmp(behavioral)
                Generate lookup__24
                        lkup: lkup_cmp(behavioral)
                Generate lookup__25
                        lkup: lkup_cmp(behavioral)
                Generate lookup__26
                        lkup: lkup_cmp(behavioral)
Package package_automata
Package parameters
Package std_logic_unsigned
Package std_logic_arith
Package std_logic_1164
Package standard
```

Appendix B – VHDL Code

==========

The following pages show the VHDL code implementation of the design.

==========

```
-- Paolo A Tamayo ECE700
-- This is the top module of the reconfigurable HFB-FSM
-- This design is for n-number of fuzzy inputs one fuzzy
output
-- m-number of states and l-number of state transitions.
-- HFB-FSM for MISO systems.

--      Purpose: Implementation of the reconfigurable HFB-
FSM
-- for MISO systems

-- History:
-- created 06/08/2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;

entity HFBFSM_top is
   port (
      fzyin    : in  FUZZY_VECTOR_BUS;  -- The actual fuzzy
inputs
      zbusin   : in  FUZZY_VECTOR;      -- Z outputs for
model building
      momstart : in  std_logic;         -- signal to start
sampling the vector (pulse only)
      clk      : in  std_logic;
      reset_l  : in  std_logic;
      op       : in  std_logic;         -- this selects the
type of operation
      convert  : in  std_logic;         -- conversion of the
fuzzy state rules
      enable   : in  std_logic;         -- enables model
building
      fstate   : in  std_logic_vector((NO_FUZZY_STATES-1)
downto 0);
      fzyout   : out fuzzy_out_inf;     -- fuzzy output
      newstrdy : out std_logic;         -- pulse to indicate
new state is available
      fzy_nxt_state : out std_logic_vector((NO_FUZZY_STATES-
1) downto 0)); -- new state selected
end HFBFSM_top;

architecture Behavioral of HFBFSM_top is

   component mibtop is
      port (
         fzy_inpts     : in FUZZY_VECTOR_BUS;  -- The fuzzy
inputs ( a collection of busses)
         momstart      : in std_logic;         -- signal to
start sampling the vector (pulse only)
         clk           : in std_logic;
         reset_l       : in std_logic;
         infenable     : out std_logic;
         newstrdy      : out std_logic;
         fzy_nxt_state : out
std_logic_vector((NO_FUZZY_STATES-1) downto 0));
   end component;

       component parallel_infsiso is
      port(
         xbusin        : in  FUZZY_VECTOR_BUS;
         zbusin        : in  FUZZY_VECTOR;
          convert       : in  std_logic;
         op            : in  std_logic;      -- this selects
the type of operation
         enable        : in  std_logic;      -- enables
model building or inference
         reset_l       : in  std_logic;
         clk           : in  std_logic;
         state         : in
std_logic_vector((NO_FUZZY_STATES-1) downto 0);
         fzyout        : out fuzzy_out_inf);
   end component;


       --signal mom_start : std_logic;
       signal ifenable  : std_logic;
       signal fenable   : std_logic;
```

```
       signal state     :
std_logic_vector((NO_FUZZY_STATES-1) downto 0);
       signal infstate  :
std_logic_vector((NO_FUZZY_STATES-1) downto 0);

begin

   inference : parallel_infsiso port map (fzyin, zbusin,
convert, op,
                                                fenable,
reset_l, clk, infstate, fzyout);
       mombfsm : mibtop port map (fzyin, momstart, clk,
reset_l, ifenable, newstrdy,
                                  state);

       fzy_nxt_state <= state;
       infstate      <= fstate when op = '1' else state;
       fenable       <= enable when op = '1' else ifenable;

end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- This is the top module of the B algorithm implementation
-- for multiple fuzzy input.  The output of this module is
the
-- next state of the Fuzzy Automata based on the present
state
-- and the sub-intervals the fuzzy inputs falls into.

--      Purpose: Implementation of the B algorithm for
-- Multiple Fuzzy Inputs

-- History:
-- created 03/22/2005
-- 04/01/2005
--   - added done signal to signify that mom computation is
finished

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;

entity mibtop is
  port (
      fzy_inpts     : in FUZZY_VECTOR_BUS;  -- The fuzzy
inputs ( a collection of busses)
      momstart      : in std_logic;         -- signal to
start sampling the vector (pulse only)
      clk           : in std_logic;
      reset_l       : in std_logic;
      infenable     : out std_logic;
      newstrdy      : out std_logic;
      fzy_nxt_state : out std_logic_vector((NO_FUZZY_STATES-
1) downto 0));
end mibtop;

architecture Structural of mibtop is

   component lkup_cmp
      port (
      crntstatus  : in std_logic_vector((LOOKUPWIDTH-
(1+NO_FUZZY_STATES)) downto 0);
      lkupentry   : in std_logic_vector((LOOKUPWIDTH-1)
downto 0);
      lkupnewst   : in std_logic;
         match       : out std_logic;
         next_state  : out std_logic_vector(NO_FUZZY_STATES-
1 downto 0));
   end component;

   component sisobtop is
      port (
         fzy_in      : in BNDRY_VALUE;         -- fuzzy
input
         fzyin_intvl : out std_logic_vector((NINTERVAL-1)
downto 0));
   end component;

   component meanofmax is
      port (
         fzyvecmem   : in FUZZY_VECTOR;         -- member of
the fuzzy input vector
         momstart    : in std_logic;            -- signal to
start sampling the vector (pulse only)
         clk         : in std_logic;
         reset_l     : in std_logic;
         momdone     : in std_logic;
         started     : in std_logic;
         setctr      : in integer range 0 to
(FUZZY_VEC_WIDTH + 3);
         done        : out std_logic;
         fzymom      : out BNDRY_VALUE);
   end component;
```

```
--=======================================================--
   -- intermediate signals
   signal fzymombus : FUZZY_INPUT_BUS;
   signal fzyintvl  : FUZZY_BDNRY_LCTN;
   signal lkentries : LOOKUP;
   signal pstate    : std_logic_vector(NO_FUZZY_STATES-1
downto 0);
   signal nstateb   : NSTATEBUS;
   signal match     : std_logic_vector(LOOKUPENTRIES-1
downto 0);
   signal nst_temp  : std_logic_vector((NO_FUZZY_STATES-1)
downto 0);
   signal prtstate  : std_logic_vector(LOOKUPWIDTH-
(1+NO_FUZZY_STATES) downto 0);
   signal done      : std_logic_vector(NO_FUZZY_INPUTS-1
downto 0);
   signal st_eval   : std_logic;

   signal started   : std_logic;
   signal momdone   : std_logic;
   signal setctr    : integer range 0 to (FUZZY_VEC_WIDTH +
3) := 0;

begin

   fzy_nxt_state <= nst_temp;      -- this is the output
(next state of the system)
   pstate        <= nst_temp;      -- copy the next state to
the present state
   infenable     <= started;       -- signal that enables
the inference module

   -- generation of the lookup table
   limits: for i in 0 to (LOOKUPENTRIES-1) generate
   lkentries(i) <= LOOKUPTABLE(i);
   end generate limits;

-- this is the generation of the MOM-sisotop pairs to
generate the
-- interval location of each of the fuzzy input vectors
   momsiso: for i in 0 to (NO_FUZZY_INPUTS-1) generate
      mom  : meanofmax port map (fzy_inpts(i), momstart,
clk, reset_l, momdone, started, setctr, done(i),
fzymombus(i));
      stop : sisobtop port map (fzymombus(i),fzyintvl(i));
   end generate momsiso;
```



```
   -- generate the bus for the present status of the
boundary locations
   -- and present state
   prsnt_state: for i in 0 to (NO_FUZZY_INPUTS-1) generate
   prtstate(((LOOKUPWIDTH-NO_FUZZY_STATES)-
(1+(i*NINTERVAL))) downto ((LOOKUPWIDTH-NO_FUZZY_STATES)-
((i+1)*NINTERVAL))) <= fzyintvl(i);
   end generate prsnt_state;
   prtstate(((LOOKUPWIDTH-NO_FUZZY_STATES)-
(1+(NO_FUZZY_INPUTS*NINTERVAL))) downto 0) <= pstate;

   lookup : for i in 0 to (LOOKUPENTRIES-1) generate
      lkup : lkup_cmp port map (prtstate,
         lkentries(i),
         st_eval,
         match(i),
         nstateb(i));
   end generate lookup;


  initialize : process(clk, reset_l)
  begin
     if (reset_l = '0') then
        nst_temp(0) <= '1';
        nst_temp((NO_FUZZY_STATES-1) downto 1) <= (others =>
'0');
        newstrdy <= '0';
        st_eval  <= '0';
        setctr <= 0;
     else
        if (clk = '1' and clk'event) then
           st_eval <= done(0);
           newstrdy <= st_eval;
           if (started = '1') then
              setctr <= setctr + 1;
           else
              setctr <= 0;
           end if;

           if (st_eval = '1') then
              for i in 0 to (LOOKUPENTRIES-1) loop
                 if (match(i) = '1') then
                    nst_temp <= nstateb(i);
                 --else
                 --   nst_temp <= nst_temp;
                 end if;
              end loop;
           end if;
        end if;
```

```
      end if;
   end process;

   count_ctrl : process(clk, reset_l)
   begin
      if (reset_l = '0') then
         started <= '0';
         momdone <= '0';
      else
         if (clk = '1' and clk'event) then
            if (momstart = '1' and started = '0') then
               started <= '1';                          --
set start flag
               momdone <= '0';
            end if;
            if (setctr = (CONV_INTEGER(FUZZY_VEC_WIDTH -
1))) then    -- we have reached the max
               started <= '0';
               momdone    <= '1';
            else
               momdone <= '0';
            end if;
         end if;
      end if;
   end process;
end Structural;
```

```
-- Paolo A Tamayo ECE700
-- Component for the lookup table implementation to decide
the next state
-- basically a comparator

-- History:
-- created 03/25/2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;
--use work.mparameters.all;
--use work.Package_Automata.all;

entity lkup_cmp is
    port (
       crntstatus  : in std_logic_vector((LOOKUPWIDTH-
(1+NO_FUZZY_STATES)) downto 0);
       lkupentry   : in std_logic_vector((LOOKUPWIDTH-1)
downto 0);
       lkupnewst   : in std_logic;
       match       : out std_logic;
       next_state  : out std_logic_vector(NO_FUZZY_STATES-1
downto 0));
end lkup_cmp;

architecture Behavioral of lkup_cmp is

   signal  match_temp : std_logic := '0';

begin

   match <= match_temp;

   compare : process(crntstatus, lkupentry, lkupnewst)
   begin
      -- this checks for match in the concatenated lookup
entry present state,
      -- lookup entry boundary locations of the fuzzy inputs
and present state
      -- and actual boundary locations of the calculated MOM
of the fuzzy inputs
      if (lkupnewst = '1') then
         if (lkupentry(LOOKUPWIDTH-1 downto NO_FUZZY_STATES)
= crntstatus) then
            match_temp <= '1';
            next_state <= lkupentry((LOOKUPWIDTH-
((NO_FUZZY_INPUTS*FUZZY_WIDTH)+NO_FUZZY_STATES))-1 downto
0);
         else
            match_temp <= '0';
            -- if no match maintain same state
            next_state <= lkupentry((LOOKUPWIDTH-
(NO_FUZZY_INPUTS*FUZZY_WIDTH)-1) downto NO_FUZZY_STATES);
         end if;
      end if;
   end process;

end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- Component for the B Algorithm implementation
-- This is the basic comparator
```

```
-- History:
-- created 03/11/2005
-- 04/01/2005
--   - added done signal to signify that mom computation is
finished

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;
use work.Package_Automata.all;

entity meanofmax is
  port (
      fzyvecmem   : in FUZZY_VECTOR;         -- member of
the fuzzy input vector
      momstart    : in std_logic;            -- signal to
start sampling the vector (pulse only)
      clk         : in std_logic;
      reset_l     : in std_logic;
      momdone     : in std_logic;
      started     : in std_logic;
      setctr      : in integer range 0 to (FUZZY_VEC_WIDTH +
3);
      done        : out std_logic;
      fzymom      : out BNDRY_VALUE);
end meanofmax;

architecture Behavioral of meanofmax is

   component DIV_UNIT is
   port (
      CLK: in std_logic;
      DIVIDEND: in STD_LOGIC_VECTOR (data_width*2 - 1 downto
0);
      DIVISOR: in STD_LOGIC_VECTOR (data_width - 1 downto
0);
      DONE: in STD_LOGIC;
      RDY: out STD_LOGIC;
      DIV_RESULT: out STD_LOGIC_VECTOR (data_width - 1
downto 0)
   );
   end component;

   signal maxcnt     : integer := 0;
   signal total      : integer := 0;
   signal momraw     : integer := 0;
   signal currmax    : FUZZY_VECTOR;
   signal dividend   : std_logic_vector(data_width*2-1
downto 0) := (others => '0');
   signal divisor    : std_logic_vector(data_width-1 downto
0) := (others => '0');
   signal div_result : std_logic_vector(data_width-1 downto
0) := (others => '0');

begin

   fzymom <= to_boundary_vector(BNDRY_SIZE, momraw);
   momraw <= CONV_INTEGER(div_result);

   divide: DIV_UNIT port map (clk, dividend, divisor,
momdone, done, div_result);

   divide_ctrl : process(clk)
   begin
      if (clk = '1' and clk'event) then
         if (setctr = (CONV_INTEGER(FUZZY_VEC_WIDTH - 1)))
then    -- we have reached the max
            dividend <= CONV_STD_LOGIC_VECTOR (total,
data_width*2);
            divisor  <= CONV_STD_LOGIC_VECTOR (maxcnt,
data_width);
         end if;
      end if;
   end process;

   count : process(clk, reset_l)
   begin
      if (reset_l = '0') then
         currmax <= (others => '0');
         maxcnt <= 1;
         total <= 0;
      else
         if (clk = '1' and clk'event) then
            if (momstart = '1') then
               currmax <= (others => '0');
            end if;
            if (started = '1') then
               if (currmax = fzyvecmem) then     -- count
the max
                  maxcnt <= maxcnt + 1;
                  total <= total + setctr;
               elsif (currmax < fzyvecmem) then
                  maxcnt <= 1;
                  currmax <= fzyvecmem;          -- copy the
new max
```

```
                total <= setctr;              -- re-
initialize the computed total
                end if;
             else
                total  <= 0;
                maxcnt <= 1;
             end if;
          end if;
       end if;
   end process;
end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- Component for the B Algorithm implementation
-- This is the basic comparator

-- History:
-- created 03/09/2005


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;
use work.Package_Automata.all;


entity sisobtop is
  port (
      fzy_in      : in BNDRY_VALUE;         -- fuzzy input
(the MOM output)
      fzyin_intvl : out std_logic_vector((NINTERVAL-1)
downto 0));
end sisobtop;

architecture Structural of sisobtop is

   component bndrygen
      port(
         bndry   : out FUZZY_BNDRY_LIMITS);
   end component;

   component fzyToBool
      port (
         fzy_in  : in BNDRY_VALUE;         -- fuzzy input
```



```
         bndrs   : in FUZZY_BNDRY_LIMITS;      -- The
boundary limit busses (this is an
                                               -- array of
busses, see package for details)
         b_out   : out BNDRY_COUNT);       -- the array of
comparator outputs, to be
                                               -- fed into a
decoder to determine in which
                                               -- interval the
value falls into
   end component;

   component fzyInEncode
      port ( braw        : in BNDRY_COUNT;
          fzyin_intvl : out std_logic_vector((NINTERVAL-1)
downto 0));
   end component;

   signal bndry : FUZZY_BNDRY_LIMITS;
   signal braw  : BNDRY_COUNT;

begin

   bndrygen_ins    : bndrygen    port map (bndry);
   fzyToBool_ins   : fzyToBool   port map (fzy_in, bndry,
braw);
   fzyInEncode_ins : fzyInEncode port map (braw,
fzyin_intvl);

end Structural;
```

```
-- Paolo A Tamayo ECE700
-- Component for the B Algorithm implementation
-- This is the basic comparator

-- History:
-- created 03/09/2005


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;
use work.Package_Automata.all;
```

```
entity bndrygen is
   port (
      bndry   : out FUZZY_BNDRY_LIMITS);      -- the
boundary limits
end bndrygen;

architecture Behavioral of bndrygen is

begin

   limits: for i in 0 to (INPUT_BNDRY_COUNT-1) generate
      bndry(i) <= BOUNDS(i);
   end generate limits;

end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- Component for the B Algorithm implementation
-- This is the fuzzy to boolean transformation
implementation
-- comparators are used to determine in which intervals the
-- fuzzy input falls into.  This is an implementation for
only
-- one fuzzy input.

-- History:
-- created 03/08/2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.parameters.all;
use work.Package_Automata.all;

entity fzyInEncode is
   port ( braw : in BNDRY_COUNT;
       fzyin_intvl : out std_logic_vector((NINTERVAL-1)
downto 0));
end fzyInEncode;

architecture Behavioral of fzyInEncode is

begin
   -- this is an implementation of a priority encoder
   process (braw)
   begin
      fzyin_intvl <= (others => '0');
      for i in braw'reverse_range loop
         if (braw(i) = '1') then
            fzyin_intvl <= CONV_STD_LOGIC_VECTOR((i+1),
NINTERVAL);
         end if;
      end loop;
   end process;
end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- Component for the B Algorithm implementation
-- This is the basic comparator

-- History:
-- created 03/18/2005
-- 03/18/05 - changed fzyvec port from FUZZY_VECTOR to
--            std_logic_vector((FUZZY_WIDTH-1) downto 0);


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;
use work.Package_Automata.all;

entity fuzzytobool is
    Port ( fzyvec   : in  std_logic_vector((FUZZY_WIDTH-1)
downto 0);
           momstart : in  std_logic;
           clk      : in  std_logic;
           reset_l  : in  std_logic;
           fzyintvl : out std_logic_vector((NINTERVAL-1)
downto 0));
end fuzzytobool;

architecture structural of fuzzytobool is

   component sisobtop
      port(
         fzy_in      : in FUZZY_VECTOR;
         fzyin_intvl : out std_logic_vector((NINTERVAL-1)
downto 0));
   end component;
```

```
   component meanofmax
      port(
         fzyvecmem   : in std_logic_vector((FUZZY_WIDTH-1)
downto 0); -- member of the fuzzy input vector
         momstart    : in std_logic;            -- signal to
start sampling the vector (pulse only)
         clk         : in std_logic;
         reset_l     : in std_logic;
         fzymom      : out FUZZY_VECTOR);
   end component;

   signal fzymom : FUZZY_VECTOR;

begin

   mom_ins     : meanofmax port map (fzyvec, momstart, clk,
reset_l, fzymom);
   sisotop_ins : sisobtop   port map (fzymom, fzyintvl);

end structural;
```

```
-- Paolo A Tamayo ECE700
-- Component for the B Algorithm implementation
-- This is the basic comparator

-- History:
-- created 03/03/2005


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;
use work.Package_Automata.all;


entity basicComp is
   port (
      fzy_in : in BNDRY_VALUE;  -- fuzzy input
      bndry  : in BNDRY_VALUE;  -- upper boundary limit
      enable : in bit;          -- enable signal
                                -- (for cascading the
modules)
      gte    : out bit;         -- greater than or equal
      intvl  : out bit);
end basicComp;
```



```
architecture Behavioral of basicComp is

begin

   compare : process(bndry, fzy_in, enable)
   begin
      -- check if enabled then evaluate the comparison
      if (enable = '1') then
         -- if less than then greater than or equal is not
true
         if (fzy_in < bndry) then
            gte <= '0';
            intvl <= '1';
         -- greater than or equal is true
         else
            gte <= '1';
            intvl <= '0';
         end if;
      else
      -- default value is always not greater than or equal
         gte <= '0';
         intvl <= '0';
      end if;
   end process;
end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- This is the parallel inference logic for SISO

--   Purpose: Parallel logic of inference using the basic
--          SISO Inference Engine and model building module

-- History:
-- created 05/24/2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;

entity parallel_infsiso is
   port(
      xbusin      : in  FUZZY_VECTOR_BUS;
      zbusin      : in  FUZZY_VECTOR;
      convert     : in  std_logic;
```

```
      op          : in  std_logic;      -- this selects the
type of operation
      enable      : in  std_logic;      -- enables model
building or inference
      reset_l     : in  std_logic;
      clk         : in  std_logic;
      state       : in  std_logic_vector((NO_FUZZY_STATES-1)
downto 0);
      fzyout      : out fuzzy_out_inf);
end parallel_infsiso;

architecture structural of parallel_infsiso is

  component basic_infsiso is
     port (
        xin         : in  FUZZY_VECTOR;
        zin         : in  FUZZY_VECTOR;
        op          : in  std_logic;      -- this selects
the type of operation
        enable      : in  std_logic;      -- this enables
the operation
        convert     : in  std_logic;
        clk         : in  std_logic;
        reset_l     : in  std_logic;
        state       : in  std_logic_vector((NO_FUZZY_STATES-
1) downto 0);
        fzyout      : out fuzzy_out_inf);
   end component;

   component multin_min_sel is
      port(
         fuzzy_bus     : in  bfuzzy_out_inf;
         fzyout        : out fuzzy_out_inf
      );
   end component;

   signal fzyout_in : bfuzzy_out_inf;

begin
   parallel: for i in 0 to (NO_FUZZY_INPUTS-1) generate
      par_inf : basic_infsiso port map(xbusin(i), zbusin,
op, enable, convert, clk, reset_l, state, fzyout_in(i));
   end generate parallel;

   min : multin_min_sel port map(fzyout_in, fzyout);

end structural;
```



```
-- Paolo A Tamayo ECE700
-- This is the basic inference logic for SISO

--   Purpose: Basic logic of inference for parallel
inference
--          of a SISO Inference Engine and model building

-- History:
-- created 05/22/2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;

entity basic_infsiso is
  port (
      xin         : in  FUZZY_VECTOR;
      zin         : in  FUZZY_VECTOR;
      op          : in  std_logic;      -- this selects the
type of operation
      enable      : in  std_logic;      -- this enables the
operation
      convert     : in  std_logic;
      clk         : in  std_logic;
      reset_l     : in  std_logic;
      state       : in  std_logic_vector((NO_FUZZY_STATES-1)
downto 0);
      fzyout      : out fuzzy_out_inf);
end basic_infsiso;

architecture Behavioral of basic_infsiso is
   signal rin       : FUZZY_VECTOR;
   signal rout      : FUZZY_VECTOR;
   signal min       : FUZZY_VECTOR;
   signal max       : FUZZY_VECTOR;

   signal ftemp     : rule_memory;

   signal tmemrule  : tmem_rule;
   signal fmemrule  : rule_mem_array;
   signal memrule   : rule_mem_array;
   signal pointer   : integer  range 0 to
(FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1 := 0;
```

```
   signal fzypnt    : integer  range 0 to
(FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1 := 0;
   signal rstate    : integer  range 0 to  NO_FUZZY_STATES-1
:= 0;

   signal dly_cnv   : std_logic;    -- delayed convert
signal
   signal tempg     : G_MTRX;


begin

   tempg <= g_matrix;

   tstate_rule_gen: process(memrule, tmemrule)
   begin
      for i in 0 to (NO_FUZZY_STATES-1) loop
         for j in 0 to (NO_FUZZY_STATES-1) loop
            for k in 0 to
((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) loop
               if (memrule(j)(k) >
g_matrix((i*NO_FUZZY_STATES)+j)) then
                  tmemrule(i)(j)(k) <=
g_matrix((i*NO_FUZZY_STATES)+j);
               else
                  tmemrule(i)(j)(k) <= memrule(j)(k);
               end if;
            end loop;
         end loop;
      end loop;
   end process;

   f_model: process(clk, reset_l)
   variable temp  : FUZZY_VECTOR;
   begin
      if (reset_l = '0') then
         for j in 0 to (NO_FUZZY_STATES-1) loop
            for k in 0 to
((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) loop
               fmemrule(j)(k) <= (others => '0');
            end loop;
          end loop;
      elsif (clk = '1' and clk'event) then

         -- delay the convert signal
         dly_cnv <= convert;

         if (convert = '1') then
         for i in 0 to (NO_FUZZY_STATES-1) loop
                  for k in 0 to
((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) loop
                  temp := (others => '0');

                     for j in 0 to (NO_FUZZY_STATES-1) loop
                        if (temp < tmemrule(i)(j)(k)) then
                           temp := tmemrule(i)(j)(k);
                        else
                           temp := temp;
                        end if;
                        fmemrule(i)(k) <= temp;
                     end loop;
                  end loop;
               end loop;
            end if;
         end if;
      end process;

      -- inference engine
      inference: process(clk, reset_l)
      variable rpointer  : integer range 0 to
(FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1;
      variable tmax      : fuzzy_out_inf;
      variable tresult   : rule_memory;
      begin
         if (reset_l = '0') then
            rpointer := 0;
            fzypnt   <= 0;
         elsif (clk = '1' and clk'event) then
            if ((enable = '1') and (op = '0')) then
               rpointer := fzypnt;
               -- this is the minimum selection
               for j in 0 to (FUZZY_VEC_WIDTH-1) loop
                  if (fmemrule(rstate)(rpointer) < xin) then
                     tresult(rpointer) :=
fmemrule(rstate)(rpointer);
                  else
                     tresult(rpointer) := xin;
                  end if;
                  if (rpointer <
((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1)) then
                     rpointer := rpointer + 1;
                  else
                     rpointer := 0;
                  end if;
               end loop;
               -- update the pointer
```

```
            if (fzypnt < ((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-
1)) then
               fzypnt <= rpointer;
            else
               fzypnt <= 0;
            end if;

            ftemp <= tresult;

         else
            -- this is the maximum selection
            for m in 0 to (FUZZY_VEC_WIDTH-1) loop
               tmax(m) := (others => '0');
               for i in 0 to (FUZZY_VEC_WIDTH-1) loop
                  if (ftemp(m+(i*FUZZY_VEC_WIDTH)) >
tmax(m)) then
                     tmax(m) :=
ftemp(m+(i*FUZZY_VEC_WIDTH));
                  end if;
               end loop;
               fzyout(m) <= tmax(m);
            end loop;
         end if;
      end if;
   end process;


   -- process to select the active rule from the rule array
   process (state)
   begin
      rstate <= 0;
      for i in state'reverse_range loop
         if (state(i) = '1') then
            rstate <= i;
         end if;
      end loop;
   end process;

   min_sel: process(zin, xin)
   begin
      if (zin < xin) then
         min <= zin;
      else
         min <= xin;
      end if;
   end process;

   max_sel: process(min, rin)
   begin
      if (min > rin) then
         max <= min;
      else
         max <= rin;
      end if;
   end process;

   clk_proc: process(clk, reset_l, pointer, memrule)
   variable wcount  : integer range 0 to
(FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1;
   begin
      pointer <= wcount;
      rin <= memrule(rstate)(pointer);
      if (reset_l = '0') then
         wcount := 0;
         -- initialize ruyle memory to contain all zero
         for i in 0 to ((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1)
loop
            for j in 0 to (NO_FUZZY_STATES-1) loop
               memrule(j)(i) <= (others => '0');
            end loop;
         end loop;

      -- op selects the operation if inference or model
building
      -- 0 - inference operation
      -- 1 - model building operation

      -- store the generated rules into the rule memory
      elsif (clk = '1' and clk'event) then
         -- convert the crisp state rules to fuzzy state
rules
         -- enble must not be asserted during this time to
         -- eliminate incomplete rule conversion, i.e. crisp
rules
         -- must be built first before the conversion to
fuzzy
         -- state rules
         if (dly_cnv = '1' and enable = '0') then
            memrule <= fmemrule;
         elsif (enable = '1') then
            if (op = '1') then
               memrule(rstate)(wcount) <= rout;
            else
               -- generate the clocked fuzzy out results
               --fzyout_in(rstate)(pointer) <= max;
            end if;
            if (wcount = ((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-
1)) then
```

```
            wcount := 0;
         else
            wcount := wcount + 1;
         end if;
      else
         wcount := 0;
      end if;
   end if;
end process;

rout <= max;
end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- This module selects the minimum value out of n-inputs

--   Purpose: Logic that implements the min selection from
--          n-inputs for a MISO configuration of the
--          inference engine

-- History:
-- created 06/07/2005

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.parameters.all;

entity multin_min_sel is
   port(
      fuzzy_bus     : in  bfuzzy_out_inf;
      fzyout        : out fuzzy_out_inf);
end multin_min_sel;

architecture Behavioral of multin_min_sel is
begin

   process (fuzzy_bus)
   variable min : fuzzy_out_inf;
   begin
      for j in (FUZZY_VEC_WIDTH-1) downto 0 loop
         min(j) := (others => '1');
         for i in 0 to (NO_FUZZY_INPUTS-1) loop
            if (fuzzy_bus(i)(j) < min(j)) then
               min(j) := fuzzy_bus(i)(j);
            end if;
```



```
         end loop;
         fzyout(j) <= min(j);
      end loop;
   end process;

end Behavioral;
```

```
-- Paolo A Tamayo ECE700
-- Package for the RECONFIGURABLE STATE TRANSITION ALGORITHM
-- FOR FUZZY AUTOMATA implementation

--   Purpose: This package defines supplemental types,
subtypes,
--   constants, and functions

-- History:
-- created 03/03/2005

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.parameters.all;

package Package_Automata is

-- Declare functions and procedure

function to_boundary_vector (size: integer; val: integer)
return BNDRY_VALUE;

end Package_Automata;

package body Package_Automata is

   function to_boundary_vector (size: integer; val: integer)
return BNDRY_VALUE is
      variable vec: BNDRY_VALUE;
      variable a: integer;

   begin
      a := val;
      for i in 0 to (size-1) loop
         if ((a mod 2) = 1) then
            vec(i) := '1';
         else
            vec(i) := '0';
```

```
         end if;
         a := a / 2;
      end loop;

      return vec;

    end to_boundary_vector;

end Package_Automata;
```

The following vhd files are the parameters files for the different examples used in testing the design.

```
-- Example 1 Parameter File
-- Paolo A Tamayo ECE700
-- Parameters for the RECONFIGURABLE STATE TRANSITION
ALGORITHM
-- FOR FUZZY AUTOMATA implementation

--   Purpose: This defines supplemental types, subtypes,
--   constants, and functions

-- History:
-- created 07/03/2005

library IEEE;
use IEEE.STD_LOGIC_1164.all;

package parameters is

     -- This is the number of Fuzzy Inputs
   constant NO_FUZZY_INPUTS  : integer := 2;

   -- This is the number of states
   constant NO_FUZZY_STATES  : integer := 3;

   -- This is the fuzzy input width (no of bits used for the
width)
   constant FUZZY_WIDTH  : integer := 4;

   -- Universal Input Set Length (or Length of the Fuzzy
Input Vector)
   constant FUZZY_VEC_WIDTH  :  integer := 7;

   -- Number of  boundaries (the number of defined
boundaries)
   constant INPUT_BNDRY_COUNT  :  integer := 3;

   -- Number of Boolean Sub-Intervals (number of intervals
the
   -- universal set is divided)
   constant NINTERVAL  :  integer := 4;

   -- this is the size of the boundary location (number of
bits needed to
   -- define a boundary)
   -- (i.e. 2^n >= Universal Input Set)
   --             where n is the value of the BNDRY_SIZE
   constant BNDRY_SIZE : integer := 4;

   --****************************************--
   -- signal types  defined
   --****************************************--

   type BNDRY_VALUE is
      array ((BNDRY_SIZE-1) downto 0) of bit;
   type FUZZY_BNDRY_LIMITS is
      array ((INPUT_BNDRY_COUNT-1) downto 0) of BNDRY_VALUE;
   type FUZZY_INPUT_BUS is
      array ((NO_FUZZY_INPUTS-1) downto 0) of BNDRY_VALUE;

   --Total Number of Transitions for the Look-Up Table
   constant Transitions : integer := 27;

   type FUZZY_VECTOR is
      array((FUZZY_WIDTH-1) downto 0) of bit;
   type BNDRY_COUNT is
      array((INPUT_BNDRY_COUNT-1) downto 0) of bit;

   -- fuzzy out data type
   type fuzzy_out_inf is
      array ((FUZZY_VEC_WIDTH-1) downto 0) of FUZZY_VECTOR;

   -- bussed fuzzy_out_inf
   type bfuzzy_out_inf is
      array ((NO_FUZZY_INPUTS-1) downto 0) of fuzzy_out_inf;

   -- data type for rule memory
   -- this creates a memory array of FUZZY_VECTOR of size
FUZZY_VEC_WIDTH
   type rule_memory is
      array(((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) downto 0)
of FUZZY_VECTOR;
```

```
   -- this creates the array of rule_memory depending on the
number of states
   type rule_mem_array is
      array(NO_FUZZY_STATES-1 downto 0) of rule_memory;

   -- this creates a matrix of rule_mem_array for fuzzy
state rule creation
   type tmem_rule is
      array(NO_FUZZY_STATES-1 downto 0) of rule_mem_array;

   -- this creates the G-MATRIX (matrix of degrees of state
membership functions)
   type G_MTRX is
      array((NO_FUZZY_STATES*NO_FUZZY_STATES)-1 downto 0) of
FUZZY_VECTOR;

   constant g_matrix : G_MTRX := (
      "1111", "0011", "1000",
      "1100", "1111", "0110",
      "0101", "1010", "1111"
      );

   type FUZZY_VECTOR_BUS is
      array((NO_FUZZY_INPUTS-1) downto 0) of FUZZY_VECTOR;

   --This is the Bundled Boundary location of Different MOMs
   type FUZZY_BDNRY_LCTN is
      array((NO_FUZZY_INPUTS-1) downto 0) of
std_logic_vector(NINTERVAL-1 downto 0);

   --This is the Boundary Points (Limits)
   constant BOUNDS : FUZZY_BNDRY_LIMITS := (
      --Upper Bound for Sub-Interval3 : 6
      "0110",
      --Upper Bound for Sub-Interval2 : 4
      "0100",
      --Upper Bound for Sub-Interval1 : 2
      "0010"
      );

   --This is the width of the Lookup Table Calculated from
sizes of
   --NO_FUZZY_INPUTS, NINTERVAL and NO_FUZZY_STATES

   --
**********************************************************
*********--
   constant data_width : integer := BNDRY_SIZE;
   constant LOOKUPWIDTH : integer :=
(NO_FUZZY_INPUTS*NINTERVAL)+(NO_FUZZY_STATES*2);
   constant LOOKUPENTRIES : integer := Transitions;

   type LOOKUP is
      array(LOOKUPENTRIES-1 downto 0) of
std_logic_vector(LOOKUPWIDTH-1 downto 0);
   type NSTATEBUS is
      array (LOOKUPENTRIES-1 downto 0) of
std_logic_vector(NO_FUZZY_STATES-1 downto 0);

   --Lookup Table which has all possible State Transitions
   constant LOOKUPTABLE : LOOKUP := (
      "00010011001010",                    -- 1
      "00010010001001",                    -- 2
      "00100001001100",                    -- 3
      "00110001001001",                    -- 4
      "00100010001100",                    -- 5
      "00010001001001",                    -- 6
      "00110010001001",                    -- 7
      "00100011001100",                    -- 8
      "00110011001001",                    -- 9

      "00010001010100",                    -- 10
      "00010010010100",                    -- 11
      "00010011010100",                    -- 12
      "00100001010010",                    -- 13
      "00100010010010",                    -- 14
      "00100011010010",                    -- 15
      "00110001010001",                    -- 16
      "00110010010001",                    -- 17
      "00110011010001",                    -- 18

      "00010001100100",                    -- 19
      "00010010100100",                    -- 20
      "00010011100100",                    -- 21
      "00100001100010",                    -- 22
      "00100010100010",                    -- 23
      "00100011100010",                    -- 24
      "00110001100001",                    -- 25
      "00110010100001",                    -- 26
      --"00110011100001"                     -- 27 (for
testing an undefined path)
      "00110010100001"                     -- 27

      );
   end parameters;

   -- this must be placed even if the contents are empty
```

```
   package body parameters is

end parameters;
```

```
-- Example 2 Parameter File


-- Paolo A Tamayo ECE700
-- Parameters for the RECONFIGURABLE STATE TRANSITION
ALGORITHM
-- FOR FUZZY AUTOMATA implementation

--   Purpose: This defines supplemental types, subtypes,
--   constants, and functions

-- History:
-- created 07/03/2005

library IEEE;
use IEEE.STD_LOGIC_1164.all;

package parameters is

   -- This is the number of Fuzzy Inputs
   constant NO_FUZZY_INPUTS  : integer := 2;

   -- This is the number of states
   constant NO_FUZZY_STATES  : integer := 3;

   -- This is the fuzzy input width (no of bits used for the
width)
   constant FUZZY_WIDTH  : integer := 4;

   -- Universal Input Set Length (or Length of the Fuzzy
Input Vector)
   constant FUZZY_VEC_WIDTH  :  integer := 7;

       -- Number of  boundaries (the number of defined
boundaries)
   constant INPUT_BNDRY_COUNT  :  integer := 3;

   -- Number of Boolean Sub-Intervals (number of intervals
the
       -- universal set is divided)
       constant NINTERVAL  :  integer := 4;

   -- this is the size of the boundary location (number of
bits needed to
   -- define a boundary)
   -- (i.e. 2^n >= Universal Input Set)
   --             where n is the value of the BNDRY_SIZE
   constant BNDRY_SIZE : integer := 4;

   --------------------------------------------------
   -- signal types  defined
   --
--------------------------------------------------

   type BNDRY_VALUE is
      array ((BNDRY_SIZE-1) downto 0) of bit;
   type FUZZY_BNDRY_LIMITS is
      array ((INPUT_BNDRY_COUNT-1) downto 0) of BNDRY_VALUE;
   type FUZZY_INPUT_BUS is
      array ((NO_FUZZY_INPUTS-1) downto 0) of BNDRY_VALUE;

   --Total Number of Transitions for the Look-Up Table
   constant Transitions : integer := 27;

   type FUZZY_VECTOR is
      array((FUZZY_WIDTH-1) downto 0) of bit;
   type BNDRY_COUNT is
      array((INPUT_BNDRY_COUNT-1) downto 0) of bit;

   -- fuzzy out data type
   type fuzzy_out_inf is
      array ((FUZZY_VEC_WIDTH-1) downto 0) of FUZZY_VECTOR;

   -- bussed fuzzy_out_inf
   type bfuzzy_out_inf is
      array ((NO_FUZZY_INPUTS-1) downto 0) of fuzzy_out_inf;

   -- data type for rule memory
   -- this creates a memory array of FUZZY_VECTOR of size
FUZZY_VEC_WIDTH
   type rule_memory is
      array(((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) downto 0)
of FUZZY_VECTOR;

   -- this creates the array of rule_memory depending on the
number of states
   type rule_mem_array is
      array(NO_FUZZY_STATES-1 downto 0) of rule_memory;

   -- this creates a matrix of rule_mem_array for fuzzy
state rule creation
```

```
   type tmem_rule is
      array(NO_FUZZY_STATES-1 downto 0) of rule_mem_array;

   -- this creates the G-MATRIX (matrix of degrees of state
membership functions)
   type G_MTRX is
      array((NO_FUZZY_STATES*NO_FUZZY_STATES)-1 downto 0) of
FUZZY_VECTOR;

   constant g_matrix : G_MTRX := (
      "1111", "0000", "0000",
      "0000", "1111", "0000",
      "0000", "0000", "1111"
      );

   type FUZZY_VECTOR_BUS is
      array((NO_FUZZY_INPUTS-1) downto 0) of FUZZY_VECTOR;

   --This is the Bundled Boundary location of Different MOMs
   type FUZZY_BDNRY_LCTN is
      array((NO_FUZZY_INPUTS-1) downto 0) of
std_logic_vector(NINTERVAL-1 downto 0);

   --This is the Boundary Points (Limits)
   constant BOUNDS : FUZZY_BNDRY_LIMITS := (
      --Upper Bound for Sub-Interval3 : 6
      "0110",
      --Upper Bound for Sub-Interval2 : 4
      "0100",
      --Upper Bound for Sub-Interval1 : 2
      "0010"
      );

   --This is the width of the Lookup Table Calculated from
sizes of
   --NO_FUZZY_INPUTS, NINTERVAL and NO_FUZZY_STATES

   --*****************************************************--
   constant data_width : integer := BNDRY_SIZE;
   constant LOOKUPWIDTH : integer :=
(NO_FUZZY_INPUTS*NINTERVAL)+(NO_FUZZY_STATES*2);
   constant LOOKUPENTRIES : integer := Transitions;

   type LOOKUP is
      array(LOOKUPENTRIES-1 downto 0) of
std_logic_vector(LOOKUPWIDTH-1 downto 0);
   type NSTATEBUS is
      array (LOOKUPENTRIES-1 downto 0) of
std_logic_vector(NO_FUZZY_STATES-1 downto 0);
```



```
   --Lookup Table which has all possible State Transitions
   constant LOOKUPTABLE : LOOKUP := (
      "00100011001001",                    -- 1
      "00010011001001",                    -- 2
      "00110011001001",                    -- 3
      "00010010001010",                    -- 4
      "00100010001010",                    -- 5
      "00110010001010",                    -- 6
      "00010001001100",                    -- 7
      "00100001001100",                    -- 8
      "00110001001100",                    -- 9

      "00100010010010",                    -- 10
      "00010010010010",                    -- 11
      "00110010010010",                    -- 12
      "00010011010001",                    -- 13
      "00100011010001",                    -- 14
      "00110011010001",                    -- 15
      "00010001010100",                    -- 16
      "00100001010100",                    -- 17
      "00110001010100",                    -- 18

      "00100001100100",                    -- 19
      "00010001100100",                    -- 20
      "00110001100100",                    -- 21
      "00010010100010",                    -- 22
      "00100010100010",                    -- 23
      "00110010100010",                    -- 24
      "00010011100001",                    -- 25
      "00100011100001",                    -- 26
      "00110011100001"                     -- 27

      );
   end parameters;

   -- this must be placed even if the contents are empty
   package body parameters is

end parameters;
```

```
-- Example 3 Parameter File


-- Paolo A Tamayo ECE700
-- Parameters for the RECONFIGURABLE STATE TRANSITION
ALGORITHM
```

```
-- FOR FUZZY AUTOMATA implementation

--   Purpose: This defines supplemental types, subtypes,
--   constants, and functions

-- History:
-- created 07/03/2005

library IEEE;
use IEEE.STD_LOGIC_1164.all;

package parameters is

     -- This is the number of Fuzzy Inputs
   constant NO_FUZZY_INPUTS  : integer := 2;

   -- This is the number of states
   constant NO_FUZZY_STATES  : integer := 12;

   -- This is the fuzzy input width (no of bits used for the
width)
   constant FUZZY_WIDTH  : integer := 4;

   -- Universal Input Set Length (or Length of the Fuzzy
Input Vector)
   constant FUZZY_VEC_WIDTH  :  integer := 101;

   -- Number of  boundaries (the number of defined
boundaries)
   constant INPUT_BNDRY_COUNT  :  integer := 4;

   -- Number of Boolean Sub-Intervals (number of intervals
the
   -- universal set is divided)
   constant NINTERVAL  :  integer := 3;

   -- this is the size of the boundary location (number of
bits needed to
   -- define a boundary)
   -- (i.e. 2^n >= Universal Input Set)
   --             where n is the value of the BNDRY_SIZE
   constant BNDRY_SIZE : integer := 8;

   --*************************************************--
   -- signal types  defined
   --*************************************************--

   type BNDRY_VALUE is
      array ((BNDRY_SIZE-1) downto 0) of bit;

   type FUZZY_BNDRY_LIMITS is
      array ((INPUT_BNDRY_COUNT-1) downto 0) of BNDRY_VALUE;
   type FUZZY_INPUT_BUS is
      array ((NO_FUZZY_INPUTS-1) downto 0) of BNDRY_VALUE;

   --Total Number of Transitions for the Look-Up Table
   constant Transitions : integer := 44;

   type FUZZY_VECTOR is
      array((FUZZY_WIDTH-1) downto 0) of bit;
   type BNDRY_COUNT is
      array((INPUT_BNDRY_COUNT-1) downto 0) of bit;

   -- fuzzy out data type
   type fuzzy_out_inf is
      array ((FUZZY_VEC_WIDTH-1) downto 0) of FUZZY_VECTOR;

   -- bussed fuzzy_out_inf
   type bfuzzy_out_inf is
      array ((NO_FUZZY_INPUTS-1) downto 0) of fuzzy_out_inf;

   -- data type for rule memory
   -- this creates a memory array of FUZZY_VECTOR of size
FUZZY_VEC_WIDTH
   type rule_memory is
      array(((FUZZY_VEC_WIDTH*FUZZY_VEC_WIDTH)-1) downto 0)
of FUZZY_VECTOR;

   -- this creates the array of rule_memory depending on the
number of states
   type rule_mem_array is
      array(NO_FUZZY_STATES-1 downto 0) of rule_memory;

   -- this creates a matrix of rule_mem_array for fuzzy
state rule creation
   type tmem_rule is
      array(NO_FUZZY_STATES-1 downto 0) of rule_mem_array;

   -- this creates the G-MATRIX (matrix of degrees of state
membership functions)
   type G_MTRX is
      array((NO_FUZZY_STATES*NO_FUZZY_STATES)-1 downto 0) of
FUZZY_VECTOR;

   constant g_matrix : G_MTRX := (
      "1111", "0000", "0000", "0000", "0000", "0000",
"0000", "0000", "0000", "0000", "0000", "0000",
      "0000", "1111", "0000", "0000", "0000", "0000",
"0000", "0000", "0000", "0000", "0000", "0000",
```

```
      "0000", "0000", "1111", "0000", "0000", "0000",
"0000", "0000", "0000", "0000", "0000", "0000",
      "0000", "0000", "0000", "1111", "0000", "0000",
"0000", "0000", "0000", "0000", "0000", "0000",
      "0000", "0000", "0000", "0000", "1111", "0000",
"0000", "0000", "0000", "0000", "0000", "0000",
      "0000", "0000", "0000", "0000", "0000", "1111",
"0000", "0000", "0000", "0000", "0000", "0000",
      "0000", "0000", "0000", "0000", "0000", "0000",
"1111", "0000", "0000", "0000", "0000", "0000",
      "0000", "0000", "0000", "0000", "0000", "0000",
"0000", "1111", "0000", "0000", "0000", "0000",
      "0000", "0000", "0000", "0000", "0000", "0000",
"0000", "0000", "1111", "0000", "0000", "0000",
      "0000", "0000", "0000", "0000", "0000", "0000",
"0000", "0000", "0000", "1111", "0000", "0000",
      "0000", "0000", "0000", "0000", "1111", "0000",
"0000", "0000", "0000", "0000", "0000", "0000",
      "0000", "0000", "0000", "0000", "0000", "0000",
"0000", "0000", "0000", "0000", "0000", "1111"
          );

   type FUZZY_VECTOR_BUS is
      array((NO_FUZZY_INPUTS-1) downto 0) of FUZZY_VECTOR;

   --This is the Bundled Boundary location of Different MOMs
   type FUZZY_BDNRY_LCTN is
      array((NO_FUZZY_INPUTS-1) downto 0) of
std_logic_vector(NINTERVAL-1 downto 0);


   constant BOUNDS : FUZZY_BNDRY_LIMITS := (
      --Upper Bound for Sub-Interval8 : 100
      "01100100",
      --Upper Bound for Sub-Interval7 : 70
      "01000110",
      --Upper Bound for Sub-Interval6 : 55
      "00110111",
      --Upper Bound for Sub-Interval5 : 40
      "00101000"
 );

   --This is the width of the Lookup Table Calculated from
sizes of
   --NO_FUZZY_INPUTS, NINTERVAL and NO_FUZZY_STATES

   --***********************************************--
   constant data_width : integer := BNDRY_SIZE;
```

```
   constant LOOKUPWIDTH : integer :=
(NO_FUZZY_INPUTS*NINTERVAL)+(NO_FUZZY_STATES*2);
   constant LOOKUPENTRIES : integer := Transitions;

   type LOOKUP is
      array(LOOKUPENTRIES-1 downto 0) of
std_logic_vector(LOOKUPWIDTH-1 downto 0);
   type NSTATEBUS is
      array (LOOKUPENTRIES-1 downto 0) of
std_logic_vector(NO_FUZZY_STATES-1 downto 0);

--Lookup Table which has all possible State Transitions
 constant LOOKUPTABLE : LOOKUP := (
"00110000000000000010000000010",
"01001100000000000010000000100",
"01001000000000000010000000100",
"01101100000000000010000000100",
"01101000000000000010000000100",
"10000100000000000010000001000",
"00110000000000000100000000001",
"01001100000000000100000000100",
"01001000000000000100000000100",
"01101100000000000100000000100",
"01101000000000000100000000100",
"10000100000000000100000001000",
"00110000000000001000000000001",
"01001100000000001000000001000",
"01001000000000001000000001000",
"01101100000000001000000001000",
"01101000000000001000000001000",
"10000100000000001000000010000",
"00110000000000010000000010000",
"01001100000000010000000100000",
"01001000000000010000000100000",
"01101100000000010000000100000",
"01101000000000010000000100000",
"10000100000000010000001000000",
"00110000000000100000000001000",
"01001100000000100000000100000",
"01001000000000100000000100000",
"01101100000000100000000100000",
"01101000000000100000000100000",
"10000100000000100000001000000",
"00110000000001000000000001000",
"01001100000001000000000100000",
"01001000000001000000000100000",
"01101100000001000000000100000",
"01101000000001000000000100000",
"10000100000001000000001000000",
"00110000000010000000000001000",
"01001100000010000000000100000",
"01001000000010000000000100000",
"01101100000010000000000100000",
"01101000000010000000000100000",
"10000100000010000000010000000",
```

```
"00110000000100000000010000000",
"00110000001000000000001000000",
"01001100000100000000100000000",
"01101000000100000000010000000",
"10000100000100000010000000000",
"10000101000000000010000000000",
"10000100010000000010000000000",
"10000100100000000010000000000"

 );
   end parameters;

   -- this must be placed even if the contents are empty
   package body parameters is

end parameters;
```

Appendix C – Testing Parameters and Results

Test Example 1 Parameters and Results

==========

The following parameters were used to test the design for a 2-input single output state transition testing and inference testing. Test Example 1.

==========

```
% STATES
S = 3;
% G MATRIX
G = [ 1.0 0.6 0.3 ;0.4 1.0 0.7 ;0.5 0.2 1.0 ];
% FUZZY INPUTS
nfi = 2;
% FUZZY OUTPUTS
nfo = 1;
% INPUT UNIVERSE OF DISCLOSURE
IU = 7;
% ELEMENTS IN INPUT UNIVERSAL SET
GIU = [ 0.0 1.0 2.0 3.0 4.0 5.0 6.0 ];

% OUTPUT UNIVERSE OF DISCLOSURE
OU = 7;
% ELEMENTS IN OUTPUT UNIVERSAL SET
GOU = [ 0.0 1.0 2.0 3.0 4.0 5.0 6.0 ];

% SUB-INTERVALS FOR INPUT UNIVERSAL SET
nds = 3;
% UPPER BOUNDARY ELEMENTS FOR SUB-INTERVALS
GSIUB = [ 2.0 4.0 6.0 ];%GSIUB[3]

% FUZZY INPUTS
IN = [ 0.0 0.0 1.0 1.0 0.5 0.0 0.0;
       0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];

% STATE TRANSITION MATRIX

% FOR FUZZY STATE1 :
S1 = 9; % STATES TRANSIENTS FROM FUZZY STATE1:
FSt11 = [ 1.0 3.0 2.0 0.0 1.0 0.0 ];
FSt12 = [ 1.0 2.0 1.0 0.0 0.0 1.0 ];
FSt13 = [ 2.0 1.0 3.0 1.0 0.0 0.0 ];
FSt14 = [ 3.0 1.0 1.0 0.0 0.0 1.0 ];
FSt15 = [ 2.0 2.0 3.0 1.0 0.0 0.0 ];
FSt16 = [ 1.0 1.0 1.0 0.0 0.0 1.0 ];
FSt17 = [ 3.0 2.0 1.0 0.0 0.0 1.0 ];
FSt18 = [ 2.0 3.0 3.0 1.0 0.0 0.0 ];
FSt19 = [ 3.0 3.0 1.0 0.0 0.0 1.0 ];

% FOR FUZZY STATE2 :
S2 = 9; % STATES TRANSIENTS FROM FUZZY STATE2:
FSt21 = [ 1.0 1.0 3.0 1.0 0.0 0.0 ];
FSt22 = [ 1.0 2.0 3.0 1.0 0.0 0.0 ];
FSt23 = [ 1.0 3.0 3.0 1.0 0.0 0.0 ];
FSt24 = [ 2.0 1.0 2.0 0.0 1.0 0.0 ];
FSt25 = [ 2.0 2.0 2.0 0.0 1.0 0.0 ];
FSt26 = [ 2.0 3.0 2.0 0.0 1.0 0.0 ];
FSt27 = [ 3.0 1.0 1.0 0.0 0.0 1.0 ];
FSt28 = [ 3.0 2.0 1.0 0.0 0.0 1.0 ];
FSt29 = [ 3.0 3.0 1.0 0.0 0.0 1.0 ];

% FOR FUZZY STATE3 :
S3 = 9; % STATES TRANSIENTS FROM FUZZY STATE3:
FSt31 = [ 1.0 1.0 3.0 1.0 0.0 0.0 ];
FSt32 = [ 1.0 2.0 3.0 1.0 0.0 0.0 ];
FSt33 = [ 1.0 3.0 3.0 1.0 0.0 0.0 ];
FSt34 = [ 2.0 1.0 2.0 0.0 1.0 0.0 ];
FSt35 = [ 2.0 2.0 2.0 0.0 1.0 0.0 ];
FSt36 = [ 2.0 3.0 2.0 0.0 1.0 0.0 ];
FSt37 = [ 3.0 1.0 1.0 0.0 0.0 1.0 ];
FSt38 = [ 3.0 2.0 1.0 0.0 0.0 1.0 ];
FSt39 = [ 3.0 3.0 1.0 0.0 0.0 1.0 ];

% BUILDING RULES

% FOR CRISP STATE1 :
RULES1 = 5;
% ************BUILDING RULE1***************

% FOR FUZZY INPUT1 :
RIN111 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN112 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT111 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
```

```
% ************BUILDING RULE2****************

% FOR FUZZY INPUT1 :
RIN121 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN122 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY OUTPUT1 :
ROUT121 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];

% ************BUILDING RULE3****************

% FOR FUZZY INPUT1 :
RIN131 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY INPUT2 :
RIN132 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT131 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];

% ************BUILDING RULE4****************

% FOR FUZZY INPUT1 :
RIN141 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY INPUT2 :
RIN142 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY OUTPUT1 :
ROUT141 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];

% ************BUILDING RULE5****************

% FOR FUZZY INPUT1 :
RIN151 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN152 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT151 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];

% FOR CRISP STATE2 :
RULES2 = 5;
% ************BUILDING RULE1****************

% FOR FUZZY INPUT1 :
RIN211 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY INPUT2 :
RIN212 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT211 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];

% ************BUILDING RULE2****************

% FOR FUZZY INPUT1 :
RIN221 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN222 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT221 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];

% ************BUILDING RULE3****************

% FOR FUZZY INPUT1 :
RIN231 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN232 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT231 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];

% ************BUILDING RULE4****************

% FOR FUZZY INPUT1 :
RIN241 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN242 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY OUTPUT1 :
ROUT241 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];

% ************BUILDING RULE5****************

% FOR FUZZY INPUT1 :
RIN251 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY INPUT2 :
RIN252 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY OUTPUT1 :
ROUT251 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];

% FOR CRISP STATE3 :
RULES3 = 5;
% ************BUILDING RULE1****************

% FOR FUZZY INPUT1 :
RIN311 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY INPUT2 :
RIN312 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT311 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];

% ************BUILDING RULE2****************

% FOR FUZZY INPUT1 :
RIN321 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
```

```
% FOR FUZZY INPUT2 :
RIN322 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY OUTPUT1 :
ROUT321 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];

% ************BUILDING RULE3****************

% FOR FUZZY INPUT1 :
RIN331 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 ];
% FOR FUZZY INPUT2 :
RIN332 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT331 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];

% ************BUILDING RULE4****************

% FOR FUZZY INPUT1 :
RIN341 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN342 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT341 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0 ];

% ************BUILDING RULE5****************

% FOR FUZZY INPUT1 :
RIN351 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY INPUT2 :
RIN352 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];
% FOR FUZZY OUTPUT1 :
ROUT351 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0 ];

% INITIAL FUZZY STATE
STATE = 1;
```

==========

The following are the generated crisp state rules and fuzzy state rules after the model building operation is performed.

==========

```
%*****************************************

 %BUILDING RULE FOR CRISP STATE :1
 %Rule :1
 %Rule :1
 R11 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0  ];
 R11 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0  ];
 R11 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 R11 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 R11 = [ 0.0 0.0 0.5 0.5 0.5 0.5 0.5  ];
 R11 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 R11 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 %Rule :2
 R12 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0  ];
 R12 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0  ];
 R12 = [ 0.5 0.5 0.5 0.5 0.5 0.0 0.0  ];
 R12 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0  ];
 R12 = [ 0.0 0.0 0.5 0.5 0.5 0.5 0.5  ];
 R12 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0  ];
 R12 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0  ];
 %BUILDING RULE FOR CRISP STATE :2
 %Rule :2
 %Rule :1
 R21 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0  ];
 R21 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0  ];
 R21 = [ 0.5 0.5 0.5 0.0 0.5 0.5 0.5  ];
 R21 = [ 1.0 1.0 0.5 0.0 0.5 1.0 1.0  ];
 R21 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 R21 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 R21 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 %Rule :2
 R22 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 R22 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 R22 = [ 0.5 0.5 0.5 0.5 0.5 0.0 0.0  ];
 R22 = [ 1.0 1.0 0.5 0.0 0.0 0.0 0.0  ];
 R22 = [ 0.5 0.5 0.5 0.0 0.5 0.5 0.5  ];
 R22 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0  ];
 R22 = [ 0.0 0.0 0.0 0.0 0.5 1.0 1.0  ];
 %BUILDING RULE FOR CRISP STATE :3
 %Rule :3
 %Rule :1
 R31 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 R31 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 R31 = [ 0.5 0.5 0.5 0.5 0.5 0.0 0.0  ];
 R31 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0  ];
 R31 = [ 0.0 0.0 0.5 0.5 0.5 0.5 0.5  ];
 R31 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 R31 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 %Rule :2
 R32 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 R32 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 R32 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 R32 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 R32 = [ 0.0 0.0 0.5 0.5 0.5 0.5 0.5  ];
 R32 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0  ];
 R32 = [ 0.0 0.0 0.5 1.0 0.5 0.0 0.0  ];

%*****************************************

%*****************************************

 %BUILDING RULE FOR FUZZY STATE :1
 %FRule :1
 RF11 = [ 1.0 1.0 0.5 0.3 0.3 0.0 0.0  ];
 RF11 = [ 1.0 1.0 0.5 0.3 0.3 0.0 0.0  ];
 RF11 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF11 = [ 0.6 0.6 0.5 1.0 0.5 1.0 1.0  ];
 RF11 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF11 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 RF11 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 %FRule :2
 RF12 = [ 1.0 1.0 0.5 0.6 0.5 0.6 0.6  ];
 RF12 = [ 1.0 1.0 0.5 0.6 0.5 0.6 0.6  ];
 RF12 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF12 = [ 0.6 0.6 0.5 1.0 0.5 0.3 0.3  ];
 RF12 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF12 = [ 0.0 0.0 0.3 0.3 0.5 1.0 1.0  ];
 RF12 = [ 0.0 0.0 0.3 0.3 0.5 1.0 1.0  ];
 %BUILDING RULE FOR FUZZY STATE :2
 %FRule :1
 RF21 = [ 1.0 1.0 0.5 0.7 0.5 0.0 0.0  ];
 RF21 = [ 1.0 1.0 0.5 0.7 0.5 0.0 0.0  ];
 RF21 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF21 = [ 1.0 1.0 0.5 0.7 0.5 1.0 1.0  ];
 RF21 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF21 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 RF21 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 %FRule :2
 RF22 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 RF22 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
```

```
 RF22 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF22 = [ 1.0 1.0 0.5 0.7 0.5 0.7 0.7  ];
 RF22 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF22 = [ 0.0 0.0 0.5 0.7 0.5 1.0 1.0  ];
 RF22 = [ 0.0 0.0 0.5 0.7 0.5 1.0 1.0  ];
 %BUILDING RULE FOR FUZZY STATE :3
 %FRule :1
 RF31 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 RF31 = [ 1.0 1.0 0.5 1.0 0.5 0.0 0.0  ];
 RF31 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF31 = [ 0.2 0.2 0.5 1.0 0.5 0.5 0.5  ];
 RF31 = [ 0.2 0.2 0.5 0.5 0.5 0.5 0.5  ];
 RF31 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 RF31 = [ 0.0 0.0 0.5 1.0 0.5 1.0 1.0  ];
 %FRule :2
 RF32 = [ 1.0 1.0 0.5 1.0 0.5 0.2 0.2  ];
 RF32 = [ 1.0 1.0 0.5 1.0 0.5 0.2 0.2  ];
 RF32 = [ 0.5 0.5 0.5 0.5 0.5 0.5 0.5  ];
 RF32 = [ 0.2 0.2 0.5 1.0 0.5 1.0 1.0  ];
 RF32 = [ 0.2 0.2 0.5 0.5 0.5 0.5 0.5  ];
 RF32 = [ 0.0 0.0 0.5 1.0 0.5 0.5 0.5  ];
 RF32 = [ 0.0 0.0 0.5 1.0 0.5 0.5 0.5  ];

%*****************************************

=========================================
```

The following pages show the results of simulations for Test Example 1 parameters.

```
=========================================


This is the log file for the state transition test with
inference.

---------------------------------
Started Rule Building Operation
---------------------------------

Fuzzy State 1 Rule For Fuzzy Input 0
{{0} {0} {5} {5} {8} {F} {F}}
{{0} {0} {5} {5} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {F} {8} {A} {A}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {F} {8} {0} {0}}
{{F} {F} {8} {F} {8} {0} {0}}

Fuzzy State 2 Rule For Fuzzy Input 0
{{0} {0} {8} {C} {8} {F} {F}}
{{0} {0} {8} {C} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {C} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {F} {8} {0} {0}}
{{F} {F} {8} {F} {8} {0} {0}}

Fuzzy State 3 Rule For Fuzzy Input 0
{{0} {0} {8} {F} {8} {F} {F}}
{{0} {0} {8} {F} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{8} {8} {8} {F} {8} {3} {3}}
{{8} {8} {8} {8} {8} {3} {3}}
{{F} {F} {8} {F} {8} {0} {0}}
{{F} {F} {8} {F} {8} {0} {0}}

Fuzzy State 1 Rule For Fuzzy Input 1
{{0} {0} {8} {A} {8} {F} {F}}
{{0} {0} {8} {A} {8} {F} {F}}
{{5} {5} {8} {8} {8} {8} {8}}
{{5} {5} {8} {F} {8} {A} {A}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {5} {5} {0} {0}}
{{F} {F} {8} {5} {5} {0} {0}}

Fuzzy State 2 Rule For Fuzzy Input 1
{{0} {0} {8} {F} {8} {F} {F}}
{{0} {0} {8} {F} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{C} {C} {8} {C} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {C} {8} {0} {0}}
{{F} {F} {8} {C} {8} {0} {0}}

Fuzzy State 3 Rule For Fuzzy Input 1
{{0} {0} {8} {F} {8} {F} {F}}
{{0} {0} {8} {F} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {F} {8} {3} {3}}
{{8} {8} {8} {8} {8} {3} {3}}
{{8} {8} {8} {F} {8} {0} {0}}
{{8} {8} {8} {F} {8} {0} {0}}


----------------------------------------
Started State Transition Tests
----------------------------------------

----------------------------------------
Transition Test 1: State 1 -> State 1
```

```
x1=low & x2=low is z=low

Results in Transition
From: 001
 TO:  001

Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
 FI2: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0

Fuzzy outputs are:  {{5} {5} {8} {8} {8} {F} {F}}

----------------------------------------
Transition Test 2: State 1 -> State 3

x1=medium & x2=high is z=high

Results in Transition
From: 001
 TO:  100

Using the inputs:
 FI1: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF

Fuzzy outputs are:  {{F} {F} {8} {8} {8} {8} {8}}

----------------------------------------
Transition Test 3: State 3 -> State 3

x1=low & x2=high is z=medium

Results in Transition
From: 100
 TO:  100

Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF

Fuzzy outputs are:  {{8} {8} {8} {F} {8} {3} {3}}

----------------------------------------
Transition Test 4: State 3 -> State 2

x1=medium & x2=medium is z=medium

Results in Transition
From: 100
 TO:  010

Using the inputs:
 FI1: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0

Fuzzy outputs are:  {{8} {8} {8} {F} {8} {8} {8}}

----------------------------------------
Transition Test 5: State 2 -> State 2

x1=medium & x2=low is z=low

Results in Transition
From: 010
 TO:  010

Using the inputs:
 FI1: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0
 FI2: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0

Fuzzy outputs are:  {{8} {8} {8} {C} {8} {F} {F}}

----------------------------------------
Transition Test 6: State 2 -> State 1

x1=high & x2=low is z=medium

Results in Transition
From: 010
 TO:  001

Using the inputs:
 FI1: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF
 FI2: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0

Fuzzy outputs are:  {{8} {8} {8} {F} {8} {8} {8}}

----------------------------------------
Transition Test 7: State 1 -> State 2

x1=low & x2=high is z=????  Value of Z is not specified in
the rules.

Results in Transition
From: 001
 TO:  010
```

```
Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0                          Fuzzy outputs are:  {{8} {8} {8} {F} {8} {3} {3}}
 FI2: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF

Fuzzy outputs are:  {{8} {8} {8} {8} {8} {8} {8}}

----------------------------------------
Transition Test 8: State 2 -> State 3

x1=low & x2=medium is z=low

Results in Transition
From: 010
 TO:  100

Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0

Fuzzy outputs are:  {{8} {8} {8} {C} {8} {F} {F}}

----------------------------------------
Transition Test 9: State 3 -> State ????  not in path, state
should stay the same

x1=high & x2=high is z=???? not in rules

Results in Transition
From: 100
 TO:  100

Using the inputs:
 FI1: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF
 FI2: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF

Fuzzy outputs are:  {{8} {8} {8} {F} {8} {3} {3}}

----------------------------------------
Transition Test 10: State 3 -> State 1

x1=high & x2=low is z=medium

Results in Transition
From: 100
 TO:  001

Using the inputs:
 FI1: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF
 FI2: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
```

Test Example 2 Parameters and Results

==========

The following parameters were used to test the design for a 2-input single output state transition testing and inference testing. Test Example 2.

==========

```
Crane Operator Example

2-inputs
1. angle
   1. negative
   2. zero
   3. positive
2. distance
   1. far
   2. near
   3. close
output
1. power
   1. negative
   2. zero
   3. positive


5 states (based on the distance)
state 1: far
state 2: near
state 3: close

Rules:
State 1: (medium)
   [MED-HIGH-HIGH]
1. if angle is zero and distance is far
   power positive
   [LOW-HIGH-HIGH]
2. if angle is negative and distance is far
   power positive
   [HIGH-HIGH-HIGH]
3. if angle is positive and distance is far
   power positive

State 2: (close)
   [MED-MED-HIGH]
1. if angle is zero and distance is near
   power positive
   [LOW-MED-HIGH]
2. if angle is negative and distance is near
   power positive
   [HIGH-MED-LOW]
3. if angle is positive and distance is near
   power negative
   [MED-LOW-MED]
4. if angle is zero and distance is close
   power zero
   [HIGH-LOW-LOW]
5. if angle is positive and distance is close
   power negative
   [LOW-LOW-med]
6. if angle is negative and distance is close
   power zero

State 3: (zero)
   [MED-LOW-MED]
1. if angle is zero and distance is close
   power zero
   [HIGH-LOW-LOW]
2. if angle is positive and distance is close
   power negative
   [LOW-LOW-MED]
3. if angle is negative and distance is close
   power zero


State Transition: [Angle Distance Present-State Next-State]

State 1 -> ??
ST1 = [2 3 1 1]
ST2 = [1 3 1 1]
ST3 = [3 3 1 1]
ST4 = [1 2 1 2]
ST5 = [2 2 1 2]
ST6 = [3 2 1 2]
ST7 = [1 1 1 3]
ST8 = [2 1 1 3]
ST9 = [3 1 1 3]
```

```
State 2 -> ??
ST1 = [2 2 2 2]
ST2 = [1 2 2 2]
ST3 = [3 2 2 2]
ST4 = [1 3 2 1]
ST5 = [2 3 2 1]
ST6 = [3 3 2 1]
ST7 = [1 1 2 3]
ST8 = [2 1 2 3]
ST9 = [3 1 2 3]

State 3 -> ??
ST1 = [2 1 3 3]
ST2 = [1 1 3 3]
ST3 = [3 1 3 3]
ST4 = [1 2 3 2]
ST5 = [2 2 3 2]
ST6 = [3 2 3 2]
ST7 = [1 3 3 1]
ST8 = [2 3 3 1]
ST9 = [3 3 3 1]
```

==========

The following pages show the results of simulations for Test Example 2 parameters.

==========

```
This is the log file for the state transition test with
inference.

--------------------------------
Started Rule Building Operation
--------------------------------

Fuzzy State 1 Rule For Fuzzy Input 0
{{F} {F} {8} {0} {0} {0} {0}}
{{F} {F} {8} {0} {0} {0} {0}}
{{8} {8} {8} {0} {0} {0} {0}}
{{F} {F} {8} {0} {0} {0} {0}}
{{8} {8} {8} {0} {0} {0} {0}}
{{F} {F} {8} {0} {0} {0} {0}}
{{F} {F} {8} {0} {0} {0} {0}}

Fuzzy State 2 Rule For Fuzzy Input 0
{{F} {F} {8} {F} {8} {0} {0}}
{{F} {F} {8} {F} {8} {0} {0}}
{{8} {8} {8} {8} {8} {0} {0}}
{{F} {F} {8} {F} {8} {0} {0}}
{{8} {8} {8} {8} {8} {8} {8}}
{{0} {0} {0} {0} {8} {F} {F}}
{{0} {0} {0} {0} {8} {F} {F}}

Fuzzy State 3 Rule For Fuzzy Input 0
{{0} {0} {8} {F} {8} {0} {0}}
{{0} {0} {8} {F} {8} {0} {0}}
{{0} {0} {8} {8} {8} {0} {0}}
{{0} {0} {8} {F} {8} {0} {0}}
{{0} {0} {8} {8} {8} {8} {8}}
{{0} {0} {0} {0} {8} {F} {F}}
{{0} {0} {0} {0} {8} {F} {F}}

Fuzzy State 1 Rule For Fuzzy Input 1
{{0} {0} {0} {0} {0} {0} {0}}
{{0} {0} {0} {0} {0} {0} {0}}
{{0} {0} {0} {0} {0} {0} {0}}
{{8} {8} {8} {0} {0} {0} {0}}
{{F} {F} {8} {0} {0} {0} {0}}
{{F} {F} {8} {0} {0} {0} {0}}

Fuzzy State 2 Rule For Fuzzy Input 1
{{0} {0} {8} {F} {8} {F} {F}}
{{0} {0} {8} {F} {8} {F} {F}}
{{8} {8} {8} {8} {8} {8} {8}}
{{F} {F} {8} {0} {8} {F} {F}}
{{8} {8} {8} {0} {8} {8} {8}}
{{0} {0} {0} {0} {0} {0} {0}}
{{0} {0} {0} {0} {0} {0} {0}}

Fuzzy State 3 Rule For Fuzzy Input 1
{{0} {0} {8} {F} {8} {F} {F}}
{{0} {0} {8} {F} {8} {F} {F}}
{{0} {0} {8} {8} {8} {8} {8}}
{{0} {0} {0} {0} {0} {0} {0}}
{{0} {0} {0} {0} {0} {0} {0}}
{{0} {0} {0} {0} {0} {0} {0}}
{{0} {0} {0} {0} {0} {0} {0}}


----------------------------------------
Started State Transition Tests
----------------------------------------

----------------------------------------
Transition Test 1: State 1 -> State 1

x1=medium & x2=high is z=high

Results in Transition
From: 001
 TO:  001

Using the inputs:
 FI1: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF

Fuzzy outputs are:  {{F} {F} {8} {0} {0} {0} {0}}

----------------------------------------
Transition Test 2: State 1 -> State 1

x1=low & x2=high is z=high

Results in Transition
```

```
From: 001
 TO:  001

Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF

Fuzzy outputs are:  {{F} {F} {8} {0} {0} {0} {0}}

----------------------------------------
Transition Test 3: State 1 -> State 1

x1=high & x2=high is z=high

Results in Transition
From: 001
 TO:  001

Using the inputs:
 FI1: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF
 FI2: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF

Fuzzy outputs are:  {{F} {F} {8} {0} {0} {0} {0}}

----------------------------------------
Transition Test 4: State 1 -> State 2

x1=low & x2=medium is z=high

Results in Transition
From: 001
 TO:  010

Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0

Fuzzy outputs are:  {{8} {8} {8} {0} {0} {0} {0}}

----------------------------------------
Transition Test 5: State 2 -> State 2

x1=medium & x2=medium is z=high

Results in Transition
From: 010
 TO:  010

Using the inputs:
 FI1: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0

Fuzzy outputs are:  {{F} {F} {8} {8} {8} {8} {8}}

----------------------------------------
Transition Test 6: State 2 -> State 2

x1=low & x2=medium is z=high

Results in Transition
From: 010
 TO:  010

Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
 FI2: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0

Fuzzy outputs are:  {{F} {F} {8} {8} {8} {0} {0}}

----------------------------------------
Transition Test 7: State 2 -> State 2

x1=high & x2=medium is z=low

Results in Transition
From: 010
 TO:  010

Using the inputs:
 FI1: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF
 FI2: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0

Fuzzy outputs are:  {{8} {8} {8} {8} {8} {F} {F}}

----------------------------------------
Transition Test 8: State 2 -> State 3

x1=medium & x2=low is z=medium

Results in Transition
From: 010
 TO:  100

Using the inputs:
 FI1: 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0
 FI2: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0

Fuzzy outputs are:  {{8} {8} {8} {F} {8} {8} {8}}
```

```
----------------------------------------
Transition Test 9: State 3 -> State 3

x1=high & x2=low is z=low

Results in Transition
From: 100
 TO:  100

Using the inputs:
 FI1: 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF
 FI2: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0

Fuzzy outputs are:  {{0} {0} {8} {8} {8} {F} {F}}

----------------------------------------
Transition Test 10: State 3 -> State 3

x1=low & x2=low is z=medium

Results in Transition
From: 100
 TO:  100

Using the inputs:
 FI1: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0
 FI2: 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0

Fuzzy outputs are:  {{0} {0} {8} {F} {8} {0} {0}}
```

Test Example 3 Parameters and Results

Hand Eye Coordination Example - this example was used to test the
state transition only.

2-inputs
1. accuracy
   1. below average
   2. average
   3. above average
   4. excellent
2. time
   1. below average
   2. average
   3. above average
   4. excellent

12 states

(see parameters.vhd file for details)

The following are the test data for the accuracy and time inputs.
The first column indicates the time where a non-zero value starts
and continues for 20 samples. After the 20th sample, the values are
again zero. Before the nth sample (first column) the samples are
also zero.

Each trial pairs teh accuracy input and time input.

Accuracy

1. 5 0.1 0.3 0.5 0.7 0.8 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.7 0.6 0.3 0.2 0.1 0.1 0.1
2. 34 0.1 0.3 0.7 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.3 0.2 0.1
3. 72 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1 0.1
4. 1 0.1 0.3 0.5 0.7 0.8 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.7 0.6 0.3 0.2 0.1 0.1 0.1
5. 3 0.1 0.3 0.5 0.7 0.8 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.7 0.6 0.3 0.2 0.1 0.1 0.1
6. 50 0.1 0.2 0.4 0.7 0.8 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.9 0.7 0.5 0.3 0.1
7. 3 0.1 0.3 0.5 0.7 0.8 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.7 0.6 0.3 0.2 0.1 0.1 0.1
8. 75 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1 0.1
9. 5 0.1 0.3 0.5 0.7 0.8 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.7 0.6 0.3 0.2 0.1 0.1 0.1
10. 5 0.1 0.3 0.5 0.7 0.8 0.9 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.7 0.6 0.3 0.2 0.1 0.1 0.1
11. 72 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1
12. 72 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1

Time

1. 72 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1
2. 50 0.1 0.1 0.1 0.7 0.9 1.0 1.0 1.0 1.0 1.0 0.9 0.8 0.7 0.6 0.5 0.3 0.1 0.1 0.1 0.1
3. 5 0.1 0.4 0.7 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.9 0.7 0.5 0.3 0.2 0.2 0.1 0.1
4. 70 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1
5. 75 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1
6. 25 0.1 0.4 0.9 1.0 1.0 1.0 1.0 1.0 0.9 0.9 0.9 0.7 0.6 0.5 0.4 0.4 0.4 0.3 0.2 0.1
7. 75 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1
8. 0 0.1 0.4 0.7 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.9 0.7 0.5 0.3 0.2 0.2 0.1 0.1 0.1
9. 72 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1
10. 72 0.1 0.4 0.8 0.9 0.9 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.8 0.6 0.5 0.4 0.2 0.1 0.1 0.1
11. 7 0.1 0.4 0.7 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.9 0.7 0.5 0.3 0.2 0.2 0.1 0.1
12. 0 0.1 0.4 0.7 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.9 0.7 0.5 0.3 0.2 0.2 0.1 0.1

State Transition Conditions
[Interval location of Accuracy, Interval location of Time, Present State, Next State]

1. 1 4 1 2 - "001 100 000000000001 000000000010",

```
2.  2 3  1  3  - "010 011 000000000001 000000000100",
3.  2 2  1  3  - "010 010 000000000001 000000000100",
4.  3 3  1  3  - "011 011 000000000001 000000000100",
5.  3 2  1  3  - "011 010 000000000001 000000000100",
6.  4 1  1  4  - "100 001 000000000001 000000001000",
7.  1 4  2  1  - "001 100 000000000010 000000000001",
8.  2 3  2  3  - "010 011 000000000010 000000000100",

9.  2 2  2  3  - "010 010 000000000010 000000000100",
10. 3 3  2  3  - "011 011 000000000010 000000000100",
11. 3 2  2  3  - "011 010 000000000010 000000000100",
12. 4 1  2  4  - "100 001 000000000010 000000001000",
13. 1 4  3  1  - "001 100 000000000100 000000000001",
14. 2 3  3  4  - "010 011 000000000100 000000001000",
15. 2 2  3  4  - "010 010 000000000100 000000001000",
16. 3 3  3  4  - "011 011 000000000100 000000001000",

17. 3 2  3  4  - "011 010 000000000100 000000001000",
18. 4 1  3  5  - "100 001 000000000100 000000010000",
19. 1 4  4  5  - "001 100 000000001000 000000010000",
20. 2 3  4  6  - "010 011 000000001000 000000100000",
21. 2 2  4  6  - "010 010 000000001000 000000100000",
22. 3 3  4  6  - "011 011 000000001000 000000100000",
23. 3 2  4  6  - "011 010 000000001000 000000100000",
24. 4 1  4  7  - "100 001 000000001000 000001000000",

25. 1 4  5  4  - "001 100 000000010000 000000001000",
26. 2 3  5  6  - "010 011 000000010000 000000100000",
27. 4 2  5  6  - "010 010 000000010000 000000100000",
28. 3 3  5  6  - "011 011 000000010000 000000100000",
29. 3 2  5  6  - "011 010 000000010000 000000100000",
30. 4 1  5  7  - "100 001 000000010000 000001000000",
31. 1 4  6  4  - "001 100 000000100000 000000001000",
32. 2 3  6  7  - "010 011 000000100000 000001000000",

33. 2 2  6  7  - "010 010 000000100000 000001000000",
34. 3 3  6  7  - "011 011 000000100000 000001000000",
35. 3 2  6  7  - "011 010 000000100000 000001000000",
36. 4 1  6  8  - "100 001 000000100000 000010000000",
37. 1 4  7  8  - "001 100 000001000000 000010000000",
38. 1 4  8  7  - "001 100 000010000000 000001000000",
39. 2 3  7  9  - "010 011 000001000000 000100000000",
40. 3 2  7  10 - "011 010 000001000000 001000000000",

41. 4 1  7  11 - "100 001 000001000000 010000000000",
42. 4 1  11 12 - "100 001 010000000000 100000000000",
43. 4 1  9  12 - "100 001 000100000000 100000000000",
44. 4 1  10 12 - "100 001 001000000000 100000000000"
```

Simulation Results are shown in Figure 4.3.

Appendix D – Test Files (do files)

==========

The following pages shows the test files (do files) used to verify the hardware design.

==========

Test Example 1 Do File

```
# Some useful constants

 #                1.0  1.0  0.5  0.0  0.0  0.0  0.0
set low    [list 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0]
 #                0.0  0.0  0.5  1.0  0.5  0.0  0.0
set medium [list 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0]
 #                0.0  0.0  0.0  0.0  0.5  1.0  1.0
set high   [list 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF]

restart

# set up log file for simulation results

set f [open "stateresult.txt" {WRONLY CREAT TRUNC}]
puts $f "This is the log file for the state transition test
with inference."

puts $f ""
puts $f "--------------------------------"
puts $f "Started Rule Building Operation"
puts $f "--------------------------------"
puts $f ""

#************************************************#

echo "#######################################################"
echo "$now ps: Start MISO RULE TEST"
echo "                         "
echo "#######################################################"

force clk      0 0, 1 5ns   -r 10ns
force reset_l  1

force fzyin(0)  4'h0
force fzyin(1)  4'h0
force zbusin    4'h0
force op        0
force convert   0
force enable    0
force fstate    001
force momstart  0

run 60 ns

force reset_l  0

run 40 ns

force reset_l  1

run 40 ns


echo "#######################################################"
echo "$now ps: Model Building - For CRISP fstate 1"
echo "                         "
echo "#######################################################"

 # fuzzy input 1 building rule
 # FIN1:    1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0

set x [concat "$low $medium $high $high $medium"]

 # fuzzy input 2 building rule
 # FIN2:    1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0

set y [concat "$low $high $medium $high $medium"]


 # fuzzy output
 # FOUT:    1.0  1.0  0.5  0.0  0.0  0.0  0.0
```

```
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0

set z [concat "$low $high $medium $high $medium"]

set i 0

force op     1
force enable 1


while {$i != 35} {

   if {$i < 7} {
      set j 0
      set k 7
   } elseif {$i < 14} {
      set j 7
      set k 14
   } elseif {$i < 21} {
      set j 14
      set k 21
   } elseif {$i < 28} {
      set j 21
      set k 28
   } elseif {$i < 35} {
      set j 28
      set k 35
   }

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $y $i]

   while {$j != $k} {
      force zbusin [lindex $z $j]

      run 10 ns
      incr j +1
   }
   incr i +1

}


force op     0
force enable 0

run 20ns


echo "#####################################################"
echo "$now ps: Model Building - For CRISP fstate 2"
echo "                          "
echo "#####################################################"

 # fuzzy input building rule
 # FIN1:    0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0

set x [concat "$high $low $medium $medium $high"]


 # fuzzy input 2 building rule
 # FIN2:    1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0

set y [concat "$low $medium $low $high $high"]

 # fuzzy output
 # FOUT:    0.0  0.0  0.5  1.0  0.5  0.0  0.0

 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0

set z [concat "$medium $low $low $high $high"]

set i 0

force fstate   010
force op       1
force enable   1


while {$i != 35} {

   if {$i < 7} {
      set j 0
```

```
      set k 7
   } elseif {$i < 14} {
      set j 7
      set k 14
   } elseif {$i < 21} {
      set j 14
      set k 21
   } elseif {$i < 28} {
      set j 21
      set k 28
   } elseif {$i < 35} {
      set j 28
      set k 35
   }

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $y $i]

   while {$j != $k} {
      force zbusin [lindex $z $j]

      run 10 ns
      incr j +1
   }
   incr i +1

}


force op     0
force enable 0

run 20ns

echo "#####################################################"
echo "$now ps: Model Building - For CRISP fstate 3"
echo "                        "
echo "#####################################################"

 # fuzzy input building rule
 # FIN1:    0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0

set x [concat "$high $low $high $low $medium"]

 # fuzzy input 2 building rule
 # FIN2:    0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          0.0  0.0  0.0  0.0  0.5  1.0  1.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0

set y [concat "$medium $high $low $low $medium"]


 # fuzzy output
 # FOUT:    0.0  0.0  0.0  0.0  0.5  1.0  1.0

 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0
 #          1.0  1.0  0.5  0.0  0.0  0.0  0.0
 #          0.0  0.0  0.5  1.0  0.5  0.0  0.0

set z [concat "$high $medium $medium $low $medium"]

set i 0

force fstate 100
force op     1
force enable 1


while {$i != 35} {

   if {$i < 7} {
      set j 0
      set k 7
   } elseif {$i < 14} {
      set j 7
      set k 14
   } elseif {$i < 21} {
      set j 14
      set k 21
   } elseif {$i < 28} {
      set j 21
      set k 28
   } elseif {$i < 35} {
      set j 28
      set k 35
   }

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $y $i]

   while {$j != $k} {
```

```
      force zbusin [lindex $z $j]

      run 10 ns
      incr j +1
   }
   incr i +1

}


force op     0
force enable 0

run 20ns

echo "#######################################################"
echo "$now ps: Model Building - For FUZZY fstates"
echo "                          "
echo "#######################################################"

force op       0
force enable   0
force convert  1

run 10ns

force convert  0

# record the computed fuzzy state rules into the log file

puts $f "Fuzzy State 1 Rule For Fuzzy Input 0"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 2 Rule For Fuzzy Input 0"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 3 Rule For Fuzzy Input 0"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(48:42)
]]
```

```
puts $f ""
puts $f "Fuzzy State 1 Rule For Fuzzy Input 1"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 2 Rule For Fuzzy Input 1"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 3 Rule For Fuzzy Input 1"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(48:42)
]]


run 100ns

#=========================================================

force op        0
force convert   0
force enable    0
force fstate    001


echo "#######################################################"
echo "$now ps: Start MISO State Transition Test"
echo "                          "
echo "#######################################################"

force clk     0 0, 1 5ns   -r 10ns
force momstart 0

run 20 ns

force fzyin(0)  4'h0
force fzyin(1)  4'h0

run 20 ns


echo "#######################################################"
```

```
echo "$now ps: Create file to store the results and read"
echo "         the initial state of the system          "
echo "##################################################"

echo "##################################################"
echo "$now ps: Test set 1"
echo " x1=low & x2=low is z=low "
echo " state 1 to state 1 "
echo "##################################################"


puts $f ""
puts $f ""
puts $f "----------------------------------------"
puts $f "Started State Transition Tests"
puts $f "----------------------------------------"
puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 1: State 1 -> State 1"
puts $f ""
puts $f "x1=low & x2=low is z=low"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $low

 # fuzzy input 2
set z $low

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}
```

```
while {[exam newstrdy] != 1} {
       run 10 ns
}

puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"

puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"



run 20 ns

#************************************************#

echo "##################################################"
echo "$now ps: Test set 2"
echo " x1=medium & x2=high is z=high "
echo " state 1 to state 3 "
echo "##################################################"

puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 2: State 1 -> State 3"
puts $f ""
puts $f "x1=medium & x2=high is z=high"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]



force momstart 1
run 10 ns
force momstart 0
```

```
set i 0

 # fuzzy input 1
set x $medium

 # fuzzy input 2
set z $high

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#*****************************************************#

echo "#########################################################"
echo "$now ps: Test set 3"
echo " x1=low & x2=high is z=medium "
echo " state 3 to state 3 "
echo "#########################################################"
```

```
puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 3: State 3 -> State 3"
puts $f ""
puts $f "x1=low & x2=high is z=medium"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $low

 # fuzzy input 2
set z $high

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"
```

```
puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#************************************************#

echo "#####################################################"
echo "$now ps: Test set 4"
echo " x1=medium & x2=medium is z=medium "
echo " state 3 to state 2 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 4: State 3 -> State 2"
puts $f ""
puts $f "x1=medium & x2=medium is z=medium"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $medium

 # fuzzy input 2
set z $medium

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}
```

```
while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#************************************************#

echo "#####################################################"
echo "$now ps: Test set 5"
echo " x1=medium & x2=low is z=low "
echo " state 2 to state 2 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 5: State 2 -> State 2"
puts $f ""
puts $f "x1=medium & x2=low is z=low"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0
```

```
 # fuzzy input 1
set x $medium

 # fuzzy input 2
set z $low

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#************************************************#

echo "#######################################################"
echo "$now ps: Test set 6"
echo " x1=high & x2=low is z=medium "
echo " state 2 to state 1 "
echo "#######################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 6: State 2 -> State 1"
puts $f ""
puts $f "x1=high & x2=low is z=medium"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $high

 # fuzzy input 2
set z $low

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"
```

```
run 20 ns

#*************************************************#

echo "#####################################################"
echo "$now ps: Test set 7"
echo " x1=low & x2=high is z=???? "
echo " state 1 to state 2 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 7: State 1 -> State 2"
puts $f ""
puts  $f  "x1=low  &  x2=high  is  z=????   Value  of  Z  is  not
specified in the rules."
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $low

 # fuzzy input 2
set z $high

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#*************************************************#

echo "#####################################################"
echo "$now ps: Test set 8"
echo " x1=low & x2=medium is z=low "
echo " state 2 to state 3 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 8: State 2 -> State 3"
puts $f ""
puts $f "x1=low & x2=medium is z=low "
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $low
```

```
 # fuzzy input 2
set z $medium

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#*************************************************#

echo "#######################################################"
echo "$now ps: Test set 9"
echo " x1=high & x2=high is z=???? "
echo " state 3 to state ???? not in path "
echo "#######################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 9: State 3 -> State ????  not in
path, state should stay the same"
puts $f ""
puts $f "x1=high & x2=high is z=???? not in rules "
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $high

 # fuzzy input 2
set z $high

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns
```

```
#************************************************#

echo "#######################################################"
echo "$now ps: Test set 10"
echo " x1=high & x2=low is z=medium "
echo " state 3 to state 1"
echo "#######################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 10: State 3 -> State 1"
puts $f ""
puts $f "x1=high & x2=low is z=medium "
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $high

 # fuzzy input 2
set z $low

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"

puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#************************************************#
close $f
```

Test Example 2 Do File

```
# Some useful constants

 #                1.0  1.0  0.5  0.0  0.0  0.0  0.0
set low    [list 4'hF 4'hF 4'h8 4'h0 4'h0 4'h0 4'h0]
 #                0.0  0.0  0.5  1.0  0.5  0.0  0.0
set medium [list 4'h0 4'h0 4'h8 4'hF 4'h8 4'h0 4'h0]
 #                0.0  0.0  0.0  0.0  0.5  1.0  1.0
set high   [list 4'h0 4'h0 4'h0 4'h0 4'h8 4'hF 4'hF]

restart

# set up log file for simulation results

set f [open "stateresult.txt" {WRONLY CREAT TRUNC}]
puts $f "This is the log file for the state transition test
with inference."

puts $f ""
puts $f "--------------------------------"
puts $f "Started Rule Building Operation"
puts $f "--------------------------------"
puts $f ""

#=========================================================
echo "#####################################################"
echo "$now ps: Start MISO RULE TEST"
echo "                          "
echo "#####################################################"

force clk      0 0, 1 5ns   -r 10ns
force reset_l  1

force fzyin(0)  4'h0
force fzyin(1)  4'h0
force zbusin    4'h0
force op        0
force convert   0
force enable    0
force fstate    001
force momstart  0

run 60 ns

force reset_l  0

run 40 ns

force reset_l  1

run 40 ns


echo "#####################################################"
echo "$now ps: Model Building - For CRISP fstate 1"
echo "                          "
echo "#####################################################"

 # fuzzy input 1 building rule

set x [concat "$medium $low $high"]

 # fuzzy input 2 building rule

set y [concat "$high $high $high"]


 # fuzzy output

set z [concat "$high $high $high"]

set i 0

force op     1
force enable 1


while {$i != 21} {

   if {$i < 7} {
      set j 0
      set k 7
   } elseif {$i < 14} {
      set j 7
      set k 14
   } elseif {$i < 21} {
      set j 14
      set k 21
   }

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $y $i]
```

```
   while {$j != $k} {
      force zbusin [lindex $z $j]

      run 10 ns
      incr j +1
   }
   incr i +1

}


force op     0
force enable 0

run 20ns


echo "#####################################################"
echo "$now ps: Model Building - For CRISP fstate 2"
echo "                         "
echo "#####################################################"

 # fuzzy input building rule

set x [concat "$medium $low $high $medium $high $low"]


 # fuzzy input 2 building rule

set y [concat "$medium $medium $medium $low $low $low"]

 # fuzzy output

set z [concat "$high $high $low $medium $low $medium"]

set i 0

force fstate   010
force op       1
force enable   1

while {$i != 42} {

   if {$i < 7} {
      set j 0
      set k 7
   } elseif {$i < 14} {
      set j 7
      set k 14
   } elseif {$i < 21} {
      set j 14
      set k 21
   } elseif {$i < 28} {
      set j 21
      set k 28
   } elseif {$i < 35} {
      set j 28
      set k 35
   } elseif {$i < 42} {
      set j 35
      set k 42
   }


   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $y $i]

   while {$j != $k} {
      force zbusin [lindex $z $j]

      run 10 ns
      incr j +1
   }
   incr i +1

}


force op     0
force enable 0

run 20ns

echo "#####################################################"
echo "$now ps: Model Building - For CRISP fstate 3"
echo "                         "
echo "#####################################################"

 # fuzzy input building rule

set x [concat "$medium $high $low"]

 # fuzzy input 2 building rule

set y [concat "$low $low $low"]


 # fuzzy output
```

```
set z [concat "$medium $low $medium"]

set i 0

force fstate 100
force op     1
force enable 1


while {$i != 21} {

   if {$i < 7} {
      set j 0
      set k 7
   } elseif {$i < 14} {
      set j 7
      set k 14
   } elseif {$i < 21} {
      set j 14
      set k 21
   }

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $y $i]

   while {$j != $k} {
      force zbusin [lindex $z $j]

      run 10 ns
      incr j +1
   }
   incr i +1

}


force op     0
force enable 0

run 20ns

echo "#####################################################"
echo "$now ps: Model Building - For FUZZY fstates"
echo "                         "
echo "#####################################################"

force op       0
force enable   0

force convert  1

run 10ns

force convert  0

# record the computed fuzzy state rules into the log file

puts $f "Fuzzy State 1 Rule For Fuzzy Input 0"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(0)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 2 Rule For Fuzzy Input 0"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(41:35)
]]
```

```
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(1)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 3 Rule For Fuzzy Input 0"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__0/par_inf/fmemrule(2)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 1 Rule For Fuzzy Input 1"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(0)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 2 Rule For Fuzzy Input 1"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(1)(48:42)
]]

puts $f ""
puts $f "Fuzzy State 3 Rule For Fuzzy Input 1"
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(6:0)]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(13:7)]
]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(20:14)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(27:21)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(34:28)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(41:35)
]]
puts          $f          [concat          [exam          -hex
/hfbfsm_top/inference/parallel__1/par_inf/fmemrule(2)(48:42)
]]


run 100ns
```

```
#=========================================================

force op        0
force convert   0
force enable    0
force fstate    001


echo "#######################################################"
echo "$now ps: Start MISO State Transition Test"
echo "                         "
echo "#######################################################"

force clk     0 0, 1 5ns   -r 10ns
force momstart 0

run 20 ns

force fzyin(0)  4'h0
force fzyin(1)  4'h0

run 20 ns


echo "#######################################################"
echo "$now ps: Create file to store the results and read"
echo "         the initial state of the system          "
echo "#######################################################"

echo "#######################################################"
echo "$now ps: Test set 1"
echo " x1=medium & x2=high is z=high "
echo " state 1 to state 1 "
echo "#######################################################"


puts $f ""
puts $f ""
puts $f "----------------------------------------"
puts $f "Started State Transition Tests"
puts $f "----------------------------------------"
puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 1: State 1 -> State 1"
puts $f ""
puts $f "x1=medium & x2=high is z=high"
puts $f ""
puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $medium

 # fuzzy input 2
set z $high

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"

puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"



run 20 ns
```

```
#=======================================================

echo "#######################################################"
echo "$now ps: Test set 2"
echo " x1=low & x2=high is z=high "
echo " state 1 to state 1 "
echo "#######################################################"

puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 2: State 1 -> State 1"
puts $f ""
puts $f "x1=low & x2=high is z=high"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]



force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $low

 # fuzzy input 2
set z $high

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#=======================================================

echo "#######################################################"
echo "$now ps: Test set 3"
echo " x1=high & x2=high is z=high "
echo " state 1 to state 1 "
echo "#######################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 3: State 1 -> State 1"
puts $f ""
puts $f "x1=high & x2=high is z=high"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $high

 # fuzzy input 2
```

```
set z $high

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#=========================================================

echo "#####################################################"
echo "$now ps: Test set 4"
echo " x1=low & x2=medium is z=high "
echo " state 1 to state 2 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 4: State 1 -> State 2"
puts $f ""
puts $f "x1=low & x2=medium is z=high"
puts $f ""
puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $low

 # fuzzy input 2
set z $medium

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#=========================================================
```

```
echo "#####################################################"
echo "$now ps: Test set 5"
echo " x1=medium & x2=medium is z=high "
echo " state 2 to state 2 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 5: State 2 -> State 2"
puts $f ""
puts $f "x1=medium & x2=medium is z=high"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $medium

 # fuzzy input 2
set z $medium

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"
puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#=========================================================


echo "#####################################################"
echo "$now ps: Test set 6"
echo " x1=low & x2=medium is z=high "
echo " state 2 to state 2 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 6: State 2 -> State 2"
puts $f ""
puts $f "x1=low & x2=medium is z=high"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $low

 # fuzzy input 2
set z $medium

while {$i != 7} {
```

```
   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#*************************************************************
*****************************#

echo "#####################################################"
echo "$now ps: Test set 7"
echo " x1=high & x2=medium is z=low "
echo " state 2 to state 2 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 7: State 2 -> State 2"
puts $f ""
puts $f "x1=high & x2=medium is z=low"
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]
```



```
force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $high

 # fuzzy input 2
set z $medium

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#=============================================================

echo "#####################################################"
echo "$now ps: Test set 8"
```

```
echo " x1=medium & x2=low is z=medium "
echo " state 2 to state 3 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 8: State 2 -> State 3"
puts $f ""
puts $f "x1=medium & x2=low is z=medium "
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $medium

 # fuzzy input 2
set z $low

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#============================================================

echo "#####################################################"
echo "$now ps: Test set 9"
echo " x1=high & x2=low is z=low "
echo " state 3 to state 3 "
echo "#####################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 9: State 3 -> State 3"
puts $f ""
puts $f "x1=high & x2=low is z=low "
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
force momstart 0

set i 0

 # fuzzy input 1
set x $high

 # fuzzy input 2
set z $low

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]
```

```
   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#=========================================================

echo "#########################################################"
echo "$now ps: Test set 10"
echo " x1=low & x2=low is z=medium "
echo " state 3 to state 3"
echo "#########################################################"


puts $f ""
puts $f "----------------------------------------"
puts $f "Transition Test 10: State 3 -> State 3"
puts $f ""
puts $f "x1=low & x2=low is z=medium "
puts $f ""

puts $f "Results in Transition"
puts -nonewline $f "From: "
puts $f [exam fzy_nxt_state]


force momstart 1
run 10 ns
```



```
force momstart 0

set i 0

 # fuzzy input 1
set x $low

 # fuzzy input 2
set z $low

while {$i != 7} {

   force fzyin(0) [lindex $x $i]
   force fzyin(1) [lindex $z $i]

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}
puts -nonewline $f " TO: "
puts $f " [exam fzy_nxt_state]"



puts $f ""
puts $f "Using the inputs:"
puts -nonewline $f " FI1:"
puts $f " $x"
puts -nonewline $f " FI2:"
puts $f " $z"

puts $f ""
puts -nonewline $f "Fuzzy outputs are: "
puts $f " [exam -hex fzyout]"

run 20 ns

#=========================================================
close $f
```

Test Example 3 Do File

```
restart

echo "#####################################################"
echo "$now ps: Start MISO Test"
echo "                          "
echo "#####################################################"

force clk     0 0, 1 5ns   -r 10ns
force reset_l 0
force momstart 0

run 20 ns

force fzy_inpts(0)  4'h0
force fzy_inpts(1)  4'h0

run 20 ns

force reset_l 1

run 10 ns

echo "#####################################################"
echo "$now ps: Create file to store the results and read"
echo "         the initial state of the system          "
echo "#####################################################"

set f [open "result.txt" {WRONLY CREAT TRUNC}]

puts $f [exam fzy_nxt_state]

close $f
echo "#####################################################"
echo "$now ps: Test set 1"
echo "           "
echo "#####################################################"


force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 5
 # accuracy: 0.1  0.3  0.5  0.7  0.8  0.9  1.0  1.0  1.0
1.0  1.0  1.0  0.8  0.7  0.6  0.3  0.2  0.1  0.1  0.1
set x [list 4'h1 4'h5 4'h8 4'hC 4'hD 4'hE 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hD 4'hC 4'hA 4'h5 4'h3 4'h1 4'h1 4'h1]

 # time starts at 72
 # time     : 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 4) && ($i < 25)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 71) && ($i < 92)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns
```

```
#*******************************************************#

echo "#######################################################"
echo "$now ps: Test set 2"
echo "          "
echo "#######################################################"


force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 34
 # accuracy: 0.1  0.3  0.7  0.8  0.9  0.9  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  0.8  0.6  0.3  0.2  0.1
set x [list 4'h1 4'h5 4'hC 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hF 4'hF 4'hF 4'hD 4'hA 4'h5 4'h3 4'h1]

 # time starts at 50
 # time     : 0.1  0.1  0.1  0.7  0.9  1.0  1.0  1.0  1.0
1.0  0.9  0.8  0.7  0.6  0.5  0.3  0.1  0.1  0.1  0.1
set z [list 4'h1 4'h1 4'h1 4'hC 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hE 4'hD 4'hC 4'hA 4'h8 4'h5 4'h1 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 33) && ($i < 54)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 49) && ($i < 70)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}
```

```
while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns

#*******************************************************#

echo "#######################################################"
echo "$now ps: Test set 3"
echo "          "
echo "#######################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 72
 # accuracy: 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set x [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

 # time starts at 5
 # time     : 0.1  0.4  0.7  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  0.9  0.7  0.5  0.3  0.2  0.2  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hC 4'hF 4'hF 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hE 4'hC 4'h8 4'h5 4'h3 4'h3 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 71) && ($i < 92)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }
```

```
   if {($i > 4) && ($i < 25)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns

#*****************************************************#

echo "#####################################################"
echo "$now ps: Test set 4"
echo "          "
echo "#####################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 1
 # accuracy: 0.1  0.3  0.5  0.7  0.8  0.9  1.0  1.0  1.0
1.0  1.0  1.0  0.8  0.7  0.6  0.3  0.2  0.1  0.1  0.1
set x [list 4'h1 4'h5 4'h8 4'hC 4'hD 4'hE 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hD 4'hC 4'hA 4'h5 4'h3 4'h1 4'h1 4'h1]

 # time starts at 70
 # time     : 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1

set z [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 0) && ($i < 21)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 69) && ($i < 90)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns


#*****************************************************#

echo "#####################################################"
echo "$now ps: Test set 5"
echo "          "
echo "#####################################################"

force momstart 1
run 10 ns
force momstart 0
```

```
set i 0
set j 0
set k 0

 # accuracy starts at 3
 # accuracy: 0.1  0.3  0.5  0.7  0.8  0.9  1.0  1.0  1.0
1.0  1.0  1.0  0.8  0.7  0.6  0.3  0.2  0.1  0.1  0.1
set x [list 4'h1 4'h5 4'h8 4'hC 4'hD 4'hE 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hD 4'hC 4'hA 4'h5 4'h3 4'h1 4'h1 4'h1]

 # time starts at 75
 # time     : 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 2) && ($i < 23)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 74) && ($i < 95)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns
```



```
#*****************************************************#

echo "#####################################################"
echo "$now ps: Test set 6"
echo "          "
echo "#####################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 50
 # accuracy: 0.1  0.2  0.4  0.7  0.8  1.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  0.9  0.7  0.5  0.3  0.1
set x [list 4'h1 4'h3 4'h6 4'hC 4'hD 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hF 4'hF 4'hF 4'hE 4'hC 4'h8 4'h5 4'h1]

 # time starts at 40
 # time     : 0.1  0.4  0.9  1.0  1.0  1.0  1.0  1.0  0.9
0.9  0.9  0.7  0.6  0.5  0.4  0.4  0.4  0.3  0.2  0.1
set z [list 4'h1 4'h6 4'hE 4'hF 4'hF 4'hF 4'hF 4'hF 4'hE
4'hE 4'hE 4'hC 4'hA 4'h8 4'h6 4'h6 4'h6 4'h5 4'h3 4'h1]

while {$i != 102} {

   if {($i > 49) && ($i < 70)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 39) && ($i < 60)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}
```

```
while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns

#*******************************************************#

echo "#######################################################"
echo "$now ps: Test set 7"
echo "          "
echo "#######################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 3
 # accuracy: 0.1  0.3  0.5  0.7  0.8  0.9  1.0  1.0  1.0
1.0  1.0  1.0  0.8  0.7  0.6  0.3  0.2  0.1  0.1  0.1
set x [list 4'h1 4'h5 4'h8 4'hC 4'hD 4'hE 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hD 4'hC 4'hA 4'h5 4'h3 4'h1 4'h1 4'h1]

 # time starts at 75
 # time     : 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hD 4'hE 4'hF 4'hF 4'hF 4'hF 4'hE
4'hE 4'hE 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 2) && ($i < 23)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 74) && ($i < 95)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns

#*******************************************************#

echo "#######################################################"
echo "$now ps: Test set 8"
echo "          "
echo "#######################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 75
 # accuracy: 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set x [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

 # time starts at 0
```

```
 # time     : 0.1  0.4  0.7  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  0.9  0.7  0.5  0.3  0.2  0.2  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hC 4'hF 4'hF 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hE 4'hC 4'h8 4'h5 4'h3 4'h3 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 74) && ($i < 95)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > -1) && ($i < 20)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns

#*****************************************************#

echo "#####################################################"
echo "$now ps: Test set 9"
echo "          "
echo "#####################################################"

force momstart 1
run 10 ns
force momstart 0
```

```
set i 0
set j 0
set k 0

 # accuracy starts at 5
 # accuracy  0.1  0.3  0.5  0.7  0.8  0.9  1.0  1.0  1.0
1.0  1.0  1.0  0.8  0.7  0.6  0.3  0.2  0.1  0.1  0.1
set x [list 4'h1 4'h5 4'h8 4'hC 4'hD 4'hE 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hD 4'hC 4'hA 4'h5 4'h3 4'h1 4'h1 4'h1]

 # time starts at 72
 # time     : 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 4) && ($i < 25)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 71) && ($i < 92)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f
```

```
run 20 ns


#*******************************************************#

echo "#######################################################"
echo "$now ps: Test set 10"
echo "          "
echo "#######################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 5
 # accuracy   0.1  0.3  0.5  0.7  0.8  0.9  1.0  1.0  1.0
1.0  1.0  1.0  0.8  0.7  0.6  0.3  0.2  0.1  0.1  0.1
set x [list 4'h1 4'h5 4'h8 4'hC 4'hD 4'hE 4'hF 4'hF 4'hF
4'hF 4'hF 4'hF 4'hD 4'hC 4'hA 4'h5 4'h3 4'h1 4'h1 4'h1]

 # time starts at 72
 # time     : 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 4) && ($i < 25)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > 71) && ($i < 92)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1
}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns

#*******************************************************#

echo "#######################################################"
echo "$now ps: Test set 11"
echo "          "
echo "#######################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 72
 # accuracy: 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set x [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

 # time starts at 7
 # time     : 0.1  0.4  0.7  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  0.9  0.7  0.5  0.3  0.2  0.2  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hC 4'hF 4'hF 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hE 4'hC 4'h8 4'h5 4'h3 4'h3 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 71) && ($i < 92)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
```

```
   }

   if {($i > 6) && ($i < 27)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns

#*****************************************************#

echo "#####################################################"
echo "$now ps: Test set 12"
echo "          "
echo "#####################################################"

force momstart 1
run 10 ns
force momstart 0

set i 0
set j 0
set k 0

 # accuracy starts at 72
 # accuracy: 0.1  0.4  0.8  0.9  0.9  1.0  1.0  1.0  1.0
1.0  1.0  0.8  0.6  0.5  0.4  0.2  0.1  0.1  0.1  0.1
set x [list 4'h1 4'h6 4'hD 4'hE 4'hE 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hD 4'hA 4'h8 4'h6 4'h3 4'h1 4'h1 4'h1 4'h1]

 # time starts at 0
 # time    : 0.1  0.4  0.7  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  0.9  0.7  0.5  0.3  0.2  0.2  0.1  0.1  0.1
set z [list 4'h1 4'h6 4'hC 4'hF 4'hF 4'hF 4'hF 4'hF 4'hF
4'hF 4'hF 4'hE 4'hC 4'h8 4'h5 4'h3 4'h3 4'h1 4'h1 4'h1]

while {$i != 102} {

   if {($i > 71) && ($i < 92)} {
      force fzy_inpts(0) [lindex $x $j]
      incr j +1
   } else {
      force fzy_inpts(0) 4'h0
   }

   if {($i > -1) && ($i < 20)} {
      force fzy_inpts(1) [lindex $z $k]
      incr k +1
   } else {
      force fzy_inpts(1) 4'h0
   }

   run 10 ns
   incr i +1

}

while {[exam newstrdy] != 1} {
       run 10 ns
}

set f [open "result.txt" {WRONLY APPEND}]

puts $f "[exam fzy_nxt_state]"

close $f


run 20 ns
```