4-2014

Design of Hardware Accelerator for Fuzzy Automata Using Dynamic Partial Reconfiguration

Chinh K. Nguyen

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Computer Engineering Commons

Recommended Citation
https://scholarworks.wmich.edu/masters_theses/477

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact wmu-scholarworks@wmich.edu.
DESIGN OF HARDWARE ACCELERATOR FOR FUZZY AUTOMATA USING DYNAMIC PARTIAL RECONFIGURATION

by

Chinh K. Nguyen

A thesis submitted to the Graduate College in partial fulfillment of the requirements for the degree of Master of Science in Engineering (Computer) Electrical and Computer Engineering Western Michigan University April 2014

Doctoral committee:

Janos L. Grantner, Ph.D., Advisor, Chair
Massood Zandi Atashbar, Ph.D.
Bradley J. Bazuin, Ph.D.
DESIGN OF HARDWARE ACCELERATOR FOR FUZZY AUTOMATA USING DYNAMIC PARTIAL RECONFIGURATION

Chinh K. Nguyen, M.S.E.

Western Michigan University, 2014

As embedded systems have become more complex and designers are required to develop new products faster while using fewer chips. FPGAs are a good choice because they offer flexibility to design on-chip devices as well as the embedded systems that are reprogrammable and reconfigurable. Today's tougher cost, higher performance, and lower power consumption requirements demand even more efficient design strategies. Dynamic partial reconfiguration (PR) is a good approach to meet these requirements because it extends the inherent flexibility of the FPGA by allowing partial regions of the FPGA to be dynamically reconfigured with new functionality while other applications are still running in the remainder of the device. Partial reconfiguration offers significant advantages compared with the traditional full reconfiguration. Designers can keep the hardware resource utilization low because they can dynamically load more logic circuits into single device in a sequential fashion. Moreover, designers can improve the productivity and scalability of their systems because they need only modify, or revise those functions that are required by a particular application [1].

In the last few decades, computational intelligence has been applied to transform human behavior and experience into mathematical representations that can be interpreted by computer programs. The development of intelligent control systems and intelligent decision support systems are part of this trend. In this Thesis, VHDL and Zynq-7000 FPGA board have been used to design all hardware modules, and a hardware accelerator of fuzzy automata – based decision support system in the context of eye-hand coordination testing for handicapped children was defined and developed.
STRUCTURE OF THESIS

This Thesis consists of five Chapters. Chapter 1 gives an overview of fuzzy systems. The first section in this Chapter introduces the basic fuzzy set and fuzzy logic concept and its application in modern systems as well as the advantages and disadvantages of fuzzy systems in practical applications. The application area involving eye-hand coordination is introduced along with the system block diagram. The FPGA Zynqboard with Processor System (PS) and Programmable Logic (PL) is also introduced in Chapter 1. Last but not least, in this chapter the partial reconfigurable function flowchart is introduced as well as the advantages of this technology in real-time applications. In Chapters 2 to 5 the details of the design and development work are discussed: the hardware, the software and the partial reconfiguration.

Chapter 2 focuses on the hardware design approach. The programmable logic design method is introduced here with detailed block diagrams, algorithms, and flowcharts of the RTL code. The detailed design and implementation of partial reconfiguration and the software application including block diagrams and flowcharts is described in Chapter 3. The first section in Chapter 3 introduces the design flow for partial reconfiguration and processor system design. All bus connections between the embedded processor and the IPs as well as the processor IP configuration are specified. The second section, which is the most important part in this Chapter, provides more information about the software design. It introduces the software interface and C program to control the dynamic partial reconfiguration features. Chapter 4 provides more details about the eye-hand coordination application, the design of the Matlab simulations, test cases, ISIM, and ChipScope waveforms. It also covers topics about fuzzified data, fuzzy state transitions and Matlab simulations design to verify the results of the real-time demonstration. The conclusions and further research plans are given in Chapter 5.
ACKNOWLEDGEMENTS

Primarily, I am grateful to my supervisor with the most sincere gratitude, Dr. Janos L. Grantner, who gave me all supports, discussions, and ideas throughout my studies. I could not have finished this without his guidance and instruction. I would like to acknowledge my Thesis committee, Dr. Massood Zandi Atashbar and Dr. Bradley J. Bazuin, who took time to help me finish my Thesis with their kind advices. I would like to say thank you to Mr. Sean Fuller, who works at Gentex Corp., and my close friend, Minh Truong, for their helps during my working Thesis time. I would like to acknowledge Mrs. Kim Tembreull, Mrs. Judy Seymour, and Mr. David Florida for their help in the graduation documents, laboratory computers, and software licenses. I would like to thank the WMU Grad College and the Electrical, and Computer Engineering Department for granting me the Graduated Assistantship during the period of time I was working on my Thesis. I am grateful to my family for their love, and support. I could not have finished my study without them. Moreover, they have given me significant motivation and encouragement for all the time I have been studying in the US.

Finally, I would like to acknowledge my friends and Professors who helped me, shared their knowledge with me, gave me advises during my study and research.

Chinh K. Nguyen
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................................................... ii

LIST OF TABLES ................................................................................................................ vi

LIST OF FIGURES ............................................................................................................... vii

NOTATIONS AND SYMBOLS ............................................................................................... ix

1. INTRODUCTION ........................................................................................................... 1

  1.1 Fuzzy control system ........................................................................................... 1

    1.1.1 Advantage and disadvantages of fuzzy control system ................................... 1

    1.1.2 Fuzzy controller architecture ........................................................................ 2

    1.1.3 Introduction of the extended Hybrid Fuzzy-Boolean Finite State Machine .... 4

    1.1.4 HFB FSM proposed hardware architecture .................................................. 6

  1.2 Eye-hand coordination ......................................................................................... 8

    1.2.1 Application problems and goals ..................................................................... 8

    1.2.2 Intelligent decision support system (IDSS) for eye-hand coordination ............ 9

  1.3 PGA Zynq-7000 development board .................................................................. 11

    1.3.1 Processing subsystem features and descriptions .......................................... 13

      1.3.1.1 Application processor unit .................................................................... 13

      1.3.1.2 Processor architecture selection ............................................................ 14

      1.3.1.3 Asymmetric multiprocessing ................................................................... 14

      1.3.1.4 Symmetric multiprocessing .................................................................... 15

    1.3.2 Operating system considerations .................................................................. 15

      1.3.2.1 Bare-metal system ............................................................................... 15

      1.3.2.2 Linux operating system ......................................................................... 16

    1.3.3 Programmable logic ...................................................................................... 16
# Table of Contents-continued

1.4 Partial reconfiguration function ................................................................. 16

2. INTRODUCTION TO THE HARDWARE SYSTEM DESIGN .................................. 20

2.1 Block RAM design ............................................................................................ 21
2.2 Mean of maxima algorithm .............................................................................. 22
2.3 Automata state transitions .............................................................................. 25
2.4 Simulation waveforms ..................................................................................... 27
2.5 ISE summary report ......................................................................................... 28

3. PARTIAL RECONFIGURATION AND SOFTWARE DESIGN ............................. 30

3.1 Partial reconfiguration design .......................................................................... 30

3.1.1 Automata transition state operation ............................................................. 31
3.1.2 Hardware interface resources ....................................................................... 31
3.1.3 File structure ................................................................................................. 32
3.1.4 PS configuration design ............................................................................... 33
3.1.5 Partial reconfiguration design flow ............................................................... 34
3.1.6 Device configuration and boot flow ............................................................... 38
3.1.7 Plan Ahead summary report .......................................................................... 40

3.2 Embedded software design flow ....................................................................... 41

3.2.1 General embedded design flow ..................................................................... 41
3.2.2 Software interface ......................................................................................... 43

4. MATLAB AND ISIM SIMULATIONS .......................................................... 45

4.1 Matlab design .................................................................................................... 45

4.1.1 Block diagram ............................................................................................... 45
4.1.2 Input data format ........................................................................................... 46
4.1.3 Create matrix function .................................................................................. 47
4.1.4 Mean of maxima module .............................................................................. 47
Table of Contents-continued

4.1.5 State transitions.........................................................................................49
4.2 Evaluation of the results ................................................................................53
   4.2.1 Definitions and input data ....................................................................53
   4.2.2 Mat lab simulation ..............................................................................56
   4.2.3 ISIM simulation ..................................................................................60
   4.2.4 ChipScope waveform .........................................................................61
5. CONCLUSIONS AND FURTHER INVESTIGATION .....................................62
REFERENCES......................................................................................................63
# LIST OF TABLES

2-1. Block RAM signals .................................................................21
2-2. Mean of maxima signals ..........................................................25
4-1. 12 States transitions conditions .................................................49
4-2. 15 States transitions conditions .................................................50
4-3. Top signal explanation..................................................................60
LIST OF FIGURES

1-1. Fuzzy controller architecture ................................................................. 3
1-2. HFB FSM model .................................................................................. 4
1-3. HFB FSM architecture ........................................................................ 7
1-4. IDSS block diagram ............................................................................. 10
1-5. IDSS flowchart ..................................................................................... 10
1-6. General Zynq-7000 block diagram ....................................................... 12
1-7. System level block diagram ................................................................ 13
1-8. Basic premise of partial reconfiguration .............................................. 17
1-9. Partial reconfiguration flow ................................................................. 18
2-1. Hardware block diagram .................................................................... 20
2-2. Distribution RAM ................................................................................ 21
2-3. MoM algorithm ................................................................................... 22
2-4. Low and high boundary of MoM value ............................................... 23
2-5. MoM block diagram ........................................................................... 24
2-6. Automata state transitions module ..................................................... 25
2-7. State transition flowchart ................................................................... 26
2-8. Automata state transition waveforms using ISIM ....................... 27
2-9. ISE summary report ............................................................................ 29
3-1. General flow of partial reconfiguration ............................................... 30
3-2. Programmable system and programmable logic connection .............. 33
3-3. Partial reconfiguration flowchart ....................................................... 34
3-4. Reconfigurable modules ................................................................... 35
3-5. RP resources required ......................................................................... 37
3-6. Physical area required ........................................................................ 37
3-7. Reconfiguration setting ...................................................................... 37
3-8. Device configuration flow including boot and reconfiguration .......... 39
3-9. Plan Ahead summary report ............................................................... 40
3-10. Floor plan report ............................................................................... 40
3-11. General software design flow .......................................................... 41
3-12. Software interface flow .................................................................... 42
3-13. User interface ................................................................................... 43
4-1. Mat lab design and block diagram ...................................................... 46
4-2. Mean of maxima flowchart ................................................................. 48
4-3. 12 states transition graph ................................................................. 51
List of Figures-continued

4-4. 15 states transition graph .................................................................52
4-5. Universal set for time and accuracy inputs ......................................54
4-6. Defuzzified time input in second ......................................................56
4-7. Defuzzified accuracy input in % .......................................................56
4-8. 12 states transitions .................................................................57
4-9. 15 states transitions .................................................................57
4-10. Defuzzified time in second ..........................................................58
4-11. Defuzzified accuracy in percent ....................................................58
4-12. 12 states transitions .................................................................59
4-13. 15 states transitions .................................................................59
4-14. ISIM simulation .................................................................60
4-15. Chip scope waveform .................................................................61
### NOTATIONS AND SYMBOLS

<table>
<thead>
<tr>
<th>NOTATION</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Average</td>
</tr>
<tr>
<td>AA</td>
<td>Above average</td>
</tr>
<tr>
<td>ACP</td>
<td>Accelerator coherency port</td>
</tr>
<tr>
<td>AMP</td>
<td>Asymmetric Multiprocessing</td>
</tr>
<tr>
<td>AP SoC</td>
<td>All Programmable System on Chip</td>
</tr>
<tr>
<td>APU</td>
<td>Application processor unit</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>AXI</td>
<td>Advanced eXensible Interface</td>
</tr>
<tr>
<td>BA</td>
<td>Below Average</td>
</tr>
<tr>
<td>BRAM</td>
<td>Block RAM</td>
</tr>
<tr>
<td>BSP</td>
<td>Board support package</td>
</tr>
<tr>
<td>CPU</td>
<td>Center Processing Unit</td>
</tr>
<tr>
<td>DDR3</td>
<td>Double Data Rate</td>
</tr>
<tr>
<td>DMAC</td>
<td>Direct Memory Access Controller</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>E</td>
<td>Excellent</td>
</tr>
<tr>
<td>FLC</td>
<td>Fuzzy Logic Controller</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FSBL</td>
<td>First Stage Boot Loader</td>
</tr>
<tr>
<td>Gb ETH</td>
<td>Giga bit Ethernet</td>
</tr>
<tr>
<td>GIC</td>
<td>General Interrupt Controller</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input/output</td>
</tr>
<tr>
<td>HFB FSM</td>
<td>Hybrid Fuzzy-Boolean Finite State Machine</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/output</td>
</tr>
<tr>
<td>IDSS</td>
<td>Intelligent Decision Support System</td>
</tr>
<tr>
<td>IOP</td>
<td>Input/Output peripherals</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Properties</td>
</tr>
<tr>
<td>MIMO</td>
<td>Multiple-input-multiple-output</td>
</tr>
<tr>
<td>MISO</td>
<td>Multiple-input-single-output</td>
</tr>
<tr>
<td>MMU</td>
<td>Memory Management Unit</td>
</tr>
<tr>
<td>MoM</td>
<td>Mean of Maxima</td>
</tr>
<tr>
<td>OC</td>
<td>Ontological Controller</td>
</tr>
<tr>
<td>OCM</td>
<td>On-Chip SRAM</td>
</tr>
<tr>
<td>OS</td>
<td>Operation system</td>
</tr>
<tr>
<td>PCAP</td>
<td>Package Capture</td>
</tr>
<tr>
<td>PL</td>
<td>Programmable logic</td>
</tr>
<tr>
<td>PR</td>
<td>Partial Reconfiguration</td>
</tr>
<tr>
<td>PS</td>
<td>Processing Subsystem</td>
</tr>
<tr>
<td>RM</td>
<td>Reconfigurable module</td>
</tr>
<tr>
<td>RP</td>
<td>Reconfigurable partition</td>
</tr>
<tr>
<td>SCU</td>
<td>Snoop Control Unit</td>
</tr>
<tr>
<td>SD</td>
<td>Secure Digital</td>
</tr>
<tr>
<td>SDK</td>
<td>Development Kit</td>
</tr>
<tr>
<td>SLCRs</td>
<td>System level control registers</td>
</tr>
<tr>
<td>SMP</td>
<td>Symmetric Multiprocessing</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>XPS</td>
<td>Xilinx Platform Studio</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

The evolution of global technologies has prompted expanding complexity of applications developed in the areas of robotics, mechanical industry, and scientific research. Imprecision and uncertainties of available information on systems are also playing important roles in increasing the complexity of applications and control systems. A few decades ago, researchers focused their efforts on creating simple and easy algorithms, using various methodologies and applications, to cope with the complexity of systems. Since then, the concept of fuzzy sets has been introduced as a means of reducing the complexity of systems due to nonlinearities, unclearly defined dynamics, the lack of information, imprecision, uncertainties, and a vague description of the system. Currently, fuzzy set theory has been widely proposed for and applied in many technical areas like engineering, medicine, management, behavioral science, and economics and so on.

1.1 Fuzzy control system

The research and application of fuzzy sets have expanded into numerous technological areas such as fuzzification of pixel intensity values and fuzzy clustering for image processing, fuzzy clustering in classification, decision making, identification and fault detection, fuzzy controllers to map expert knowledge into control systems, fuzzy modeling using expert knowledge, fuzzy optimization to solve design problems, etc.

1.1.1 Advantage and disadvantages of fuzzy control system

In general, there are five types of systems that fuzziness may prove advantages due to intentionally limited accuracy or reduced complexity to build the system and the control algorithm required [3]:

- Complex systems are difficult or impossible to model.
- Systems controlled by human experts.
- Systems use complex and continuous inputs and outputs.
- Systems use human observation for inputs or as basis for rules.
Systems are naturally vague, such as those in the behavioral and social sciences.

In complex systems, fuzzy control is a good approach because it brings some benefits that make the system simpler to design and control [3):

- The system requires fewer values, rules, decisions, and variables.
- Variables are observable and can be evaluated.
- Systems use linguistic variables that make them easier for humans to understand and monitor.
- The simplicity of the system allows the solution of previously untenable problems.
- It brings advantages for rapid prototyping because designers do not have to implement every needed feature of the system before starting verification.
- Simpler systems are cheaper to design than conventional systems.
- The robustness property of the control system may increase.

However, there are also drawbacks and challenges for those who are using or designing fuzzy controller for their systems [3].

- It is not easy to develop a model for a fuzzy system that can be analytically proven as stable.
- Because they are easier to design and faster to prototype than conventional control systems, fuzzy systems require more simulation, testing, and fine-tuning to make them more stable before they are deployed.

### 1.1.2 Fuzzy controller architecture

The fuzzy controller is a computer-based system that implements a reasoning process similar to that of human experts within a specific domain of knowledge. The purpose of this system is to utilize the experts’ knowledge, understanding, and problem solving capabilities in a particular area to control a plant. *Fig. 1.1* shows the general fuzzy controller architecture.
In this system, the fuzzy controller has four main components. The first one is the rule-base that holds the knowledge of how to control the system. The rule-base is represented by a set of rules in the form as follows:

\[
\text{If } X_{11} \text{ is } A_{11} \text{ and } \ldots \text{ and } X_{1n} \text{ is } A_{1n} \text{ then } Y \text{ is } B_1
\]

also

\[
\ldots
\]

also

\[
\text{If } X_{m1} \text{ is } A_{m1} \text{ and } \ldots \text{ and } X_{mn} \text{ is } A_{mn} \text{ then } Y \text{ is } B_m
\]

The second one is the inference mechanism (or inference engine) that decides which control rules are relevant with respect to the current inputs from the plant. Then using the knowledge base is to determine those outputs that will be passed on as inputs to the plant’s actuators. The third one is the fuzzification interface that converts the sensed analog input signals to fuzzy sets so that they can be processed by the inference engine. The final one is the defuzzification interface that converts the conclusion fuzzy sets determined by the inference engine back to analog signals as inputs for the plant’s actuators. We may conceive the fuzzy controller as an intelligent decision maker operating in real-time. It takes the process output data \( y(t) \), compares it to the reference
input \( r(t) \), and then decides what the actuator input \( u(t) \) should be to ensure that the plant will operate according to its objectives.

In order to design the fuzzy controller, the designer must collect plant data and operator’s experience on how the intelligent decision maker should act in the closed-loop system. Sometimes the relevant information can be obtained from skilled operators who are very knowledgeable about the plant and the control tasks. Most of the times, the control system designer should also study and understand the plant’s dynamics to design the set of rules for the automatic control of the system. The devised set of If Then rules are then loaded into the rule-base of the system and an inference strategy is chosen. At this point, system is ready to be tested to see if the closed-loop specifications are met [4].

1.1.3 Introduction of extended Hybrid Fuzzy-Boolean Finite State Machine

The Hybrid Fuzzy-Boolean Finite State Machine (HFB FSM) can be used to identify a problem in a supervisory control system and allow recovery from ontological de-synchronization [5]. The Ontological Controller (OC) is a contemporary supervisory control approach. It concerns about automated fault detection and identification of some particular types of faults and the recovery from those faults involving complex industrial control systems. The HFB FSM model was first introduced for OC to make a decision on possible recovery when unexpected changes or faults occur during the operation.

![Figure 1-2 HFB FSM model](image_url)
This extended HFB FSM model is implemented by a Boolean automaton based upon two-valued logic. The HFB FSM is defined by the formulas given below [5]:

\[ Z_F = X_F \circ R^* \]  
\[ R^* = G(R_S) \]  
\[ Z_C = D_F(Z_F) \]  
\[ U_B = f_u(X_B, W_B, X_T, y_B) \]  
\[ X_B = B(X_F) \]  
\[ Z_B = B(Z_F) \]  
\[ Y_B = f_y(X_B, W_B, Z_B, y_B, X_T) \]

\[ X_F, W_B, \text{ and } X_A \text{ stand for fuzzy input variables, digital input values, and analog inputs with associated threshold values, respectively. The comparator will compare the analog } X_A \text{ values with the preset threshold values. The result of this comparison (} X_T \text{) will be Boolean (digital) inputs to the HFB FSM system. } Z_F, Z_C, \text{ and } U_B \text{ stand for fuzzy, defuzzified fuzzy and two-valued (Boolean) outputs of the system, respectively. } R^* \text{ is the composite linguistic model (2), and } \circ \text{ is the operator of composition. Depending on the fuzzy control model, the designer can select out of many T-norm and T-conorm operators. In this project, we use the min and the max operators.} \]

A fuzzy state is made up of a set of crisp states. The HFB FSM stays simultaneously in each of them to a certain degree as expressed by a state membership function. There is a dominant crisp state for each fuzzy state set that the degree of the state membership function is one. Therefore, a fuzzy state is defined by its dominant crisp (Boolean) state and a state membership function [5].

\[ S_{Fk}: \quad S_k, g_{Sk} \]  

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. The matrix $G$ in (2) represents the matrix of state membership functions [5].

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

For each fuzzy state of the HFB FSM model, an $R^*$ composite linguistic model is created from the finite set of $R_{S_i}$ overall linguistic models ($i = 1 \ldots p$). If the HFB FSM is in a fuzzy state $R_{Fk}$, then we have,

$$R^*_k = \max[\min(\beta_1^k, R_{S1}), \min(\beta_2^k, R_{S1}), \ldots, \min(\beta_p^k, R_{Sp})]$$

(9)

Where $\beta_1^k \ldots \beta_p^k$ represent the degrees of state membership function ($g_{S_k}$) and $R_{S1} \ldots R_{Sp}$ are the overall linguistic models in crisp states $S_1 \ldots S_p$. New $R^*$ composite linguistic models can be created in real time by modifying the $\beta$ degrees of the state membership function on-line. $X_B$, $Z_B$, $Y_B$ and $y_B$ represent digital inputs, outputs and the next state and current state variables, in that order. The symbol $B$ indicates that variables are being converted from Fuzzy values to Boolean values in order to map a change in the status of a 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 function $D_F$ [5]. In this Thesis, we consider only a double fuzzy input and multiple digital output implementation of the HFB FSM.

1.1.4 HFB FSM proposed hardware architecture

Fig. 1.3 depicts the HFB-FSM architecture with multiple fuzzified inputs and a single output (MISO). In this Thesis, the multiple Boolean inputs and analog inputs with thresholds are omitted. Therefore, the relationship between present state and next state is as follows:

$$Y_B = fu (X_B, y_B)$$

(10)
X_B stands for sets of Boolean variables (Boolean vectors) from multiple fuzzy inputs and the y_B is the present crisp state and Y_B is the next crisp state of the automata. Further details of this model will be discussed in the hardware design section (Chapter 2). The composition module in this design is to infer the fuzzy output using the fuzzy inputs and the composite linguistic model that belongs to the current state of the automata. In other words, a composite linguistic model is created for each defined state of the automata. This operation is given by Equation (2) where R* defines the composite linguistic model and X_F represents the multiple fuzzy inputs.

In this architecture, the extended HFB FSM processes the states of the fuzzified inputs to devise the next state. The inference module of this architecture is to generate the
fuzzy output corresponding to the current state. The composite linguistic models implemented in the Rule Bank are based upon the overall linguistic model of the system and the state membership function matrix (R*). Because the fuzzy output is inferred using the observed fuzzy inputs and the current state of the system, only those contents of the rule bank are valid that are assigned to the current state. The matrix of state membership function provides a means for the system designer to tune the outputs by manipulating the elements of the matrix.

1.2 **Eye-hand coordination**

In this section, we will discuss more about the eye-hand coordination research as well as the goal of this research. Actually, this Thesis did not take part in the eye-hand coordination research project. We just use its database and transition graphs as an example for our Fuzzy Automata model.

1.2.1 **Application problems and goals**

Learning how to write plays an important role for children. Problems with acquiring proper handwriting or drawing skills frequently happen in public schools and this is the main reason why students are referred to occupational therapy services. According to research conducted in this area, a few identified elements that affect the quality of handwriting. They are as follows: kinesthesia, motor planning, eye-hand coordination, and in-hand manipulation. Children with genetic anomalies or neurological disorders can also have problems with eye-hand coordination such as children with Down syndrome, cerebral palsy and learning disabilities [6]. All these diagnoses require treatment for eye-hand coordination problems since proper eye-hand coordination skills are necessary for successfully performing daily activities of living such as taking shower, dressing, feeding, drawing, and writing. This approach is not the best if the assessment of the eye-hand coordination skills of children with disabilities and the decision making on the next, more complex test is done by using only crisp, quantitative terms. Fuzzy logic-based decision may provide a better approach because it allows the aggregation of measured data and occupational therapy expert knowledge in a common mathematical
model. A fuzzy automata model can help in developing an intelligent decision support system by suggesting the sequence and complexity of the next test to be given to the subject [5] [6]. This recommendation will depend upon the results obtained from the previously performed test and will be based upon expert knowledge. The intelligent decision making system can be fine-tuned as more test results become available.

The goal of the research was to develop an automated assessment and training procedure for children with eye-hand coordination problem. An automated assessment system is expected to reduce the burden and the associated cost of having a trained occupational therapy professional present at any individual assessment, or training session. The intelligent decision support system will be based upon a fuzzy automaton [6] [5]. In eye-hand coordination research, they have designed various experiments using robots along with visual special effects to improve the subject’s concentration on the task and to assist the subject’s motor skills.

1.2.2 Intelligent decision support system (IDSS) for eye-hand coordination

The system has two major sections. The Main Controller part includes a computer workstation, a PHANToM robot, and the software that provides for interface with user. The other important part is a fuzzy automaton that was configured to accommodate the testing of the 5-year old children group. The block diagram of the full system is shown in the Fig. 1.4. A very simple fuzzy automaton was used in the eye-hand coordination research without an inference engine. The state transition graph of the automaton was used to recommend a decision on selecting the next test case.
Depending on the age group, a designated first task will be assigned to the subject. The system then evaluates the accuracy that is the number of times the trajectory drawn by the subject is out of the desired path, and the time taken, in seconds, to execute the task. These measured values are then fuzzified and are fed as inputs to the IDSS.
Based on the current state and the evaluation of the fuzzified results of the test, the fuzzy automaton will make a transition to a new state. It will recommend the next test case that should be performed by the child. Based upon the evaluated performance, the subject may be given a less demanding test next, or he/she will be stuck at the same level, or move up to a more complex test. If the subject performs really well he/she can move to the stop state very quickly. The higher the state number of the automaton, the more difficult the test will become [6].

1.3 PGA Zynq-7000 development board

The growths in the number of fuzzy logic applications and complex customized implementations have led to the need of finding efficient and economic ways for hardware implementations. The continued advancement of Field Programmable Gate Arrays (FPGAs) devices, with their reconfigurable logic functions, portability, low power consumption, high dynamic performance of operations and large data storage capacity make them a primary candidate for developing and prototyping fuzzy logic-based embedded systems [7]. This section introduces the structure of the Zynq-7000 FPGA development board and the dynamic partial reconfiguration function that are used in this Thesis for hardware and software design.

The Zynq-7000 family is based upon the Xilinx All Programmable System on Chip (AP SoC) architecture. The Zynq-7000 FPGA chip integrates a dual-core ARM Cortex-A9 based processing system (PS) with Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 Core CPUs are the heart of the PS, which includes both on-chip memory and external memory interfaces and a set of input/output (I/O) peripherals. The Zynq-7000 introduces the flexibility and scalability of an FPGA, while providing high dynamic performance, low power, and ease of use typically associated with ASIC devices. The Zynq-7000 enables designers to save cost while allows high performance on a single platform that can be configured and programmed by using industry-standard tools. The Zynq-7000 devices are powerful to function in a wide range of applications including complex systems such as automotive driver assistance, driver information,
broadcast camera, industrial motor control, machine vision, IP and smart camera, medical diagnostics and imaging, multifunction printers, video and night vision equipment [8].

The Zynq-7000 architecture conveniently supports the design of custom logic and software in the PL and PS, respectively. The integration of the PS with the PL allows such a high level of performance that other chip solutions cannot match due to their limited I/O bandwidth, number of pins and power consumption. Moreover, with the Zynq-board designer can write the software for stand-alone and Linux OS, the libraries and drivers are available for the peripherals in the PS and the PL from Xilinx and board support packages (BSPs) as well as from partners. Fig. 1.6 shows the top-level architecture of the Zynq board [9].

Figure 1-6 General Zynq-7000 block diagram

There is the Processor Subsystem (PS) and the Programmable Logic (PL) on the Zynq chip. In addition, there are more than 3000 connections available to map signals between the PS and the PL segments. In addition, there are many peripheral IPs
connected to the PS such as DDR3 RAM, Gigabit ETH, SPI, SD card etc. The Zynq-7000 SoC is composed of the following major functional blocks: Processing System (PS), Application Processor Unit (APU), memory interfaces, I/O peripherals (IOP), Central Interconnect, and Programmable Logic (PL) [8, 9].

1.3.1 Processing subsystem features and descriptions

![System level block diagram](image)

Figure 1-7 System level block diagram

1.3.1.1 Application processor unit

The Application processing Unit (APU) is the most critical component of the system that comprises the Programmable System (PS), the Intellectual Properties (IPs) implemented in the PL, and board-level devices such as the external memories and the peripherals. The main interfaces through which the APU communicates to the rest of the system are two interfaces through the L2 cache controller and an interface to the On-Chip Memory (OCM) that is parallel to the L2 cache. All accesses from the dual Cortex-A9
MP system go through the Snoop Controller Unit (SCU) and all accesses from any other master that requires coherency with the Cortex-A9 MP system also need to be routed through the SCU using the ACP Port. All accesses that are not routed through the SCU are non-coherent with the Center Processing Unit (CPU) and software has to handle explicitly the synchronization and coherency. Accesses from the APU can target the OCM, DDR, PL slaves, or registers within the PS sub-blocks. In order to minimize the latency to the OCM, a dedicated master port from the SCU provides direct access by the processors and the ACP to the OCM, offering a latency that is even less than the L2 cache. All APU accesses to the DDR3 RAM interface are routed through the L2 cache controller. To improve the latencies of the DDR accesses, there is a dedicated master port from the L2 cache controller to the DDR3 memory controller that allows all APU-DDR3 transactions to bypass the main interconnects which is shared with the other masters [8].

1.3.1.2 Processor architecture selection

Because the Zynq-7000 SoC devices have dual-core ARM Cortex-A9 processors, in the design we must determine whether to use Asymmetric Multiprocessing (AMP) or Symmetric Multiprocessing (SMP). We have to make several architectural decisions before beginning embedded development on applications to run on the Zynq-7000 SoC. Similarly, a decision must be made for all embedded software projects regarding to which operating system(s) to be used. Next, we will discuss both the AMP and the SMP and provide an assessment of the trade-offs and concerns with each method.

1.3.1.3 Asymmetric multiprocessing

Asymmetric multiprocessing (AMP) is a processing model in which each processor in a multiple-processor system executes a different operating system image while sharing the same physical memory. Each image can be of the same operating system, but more typically, each image is a different operating system, complementing the other OS with different characteristics [8]. A full-featured operating system, such as Linux, will let us connect to the outside world through networking and user interfaces while a smaller, lightweight operating system can be more efficient with respect to memory and real-time operations.
The division of system devices (such as the UART, timer-counter, and Ethernet) between the processors is a critical element in system design. In general, most devices must be dedicated to their assigned processor. The interrupt controller designed to be shared with multiple processors. Communication between processors is a key element that allows both operating systems to be effective. It can be achieved in many different ways, including inter-processor interrupts, shared memory, and message passing [8] [10].

1.3.1.4 Symmetric multiprocessing

Symmetric multiprocessing (SMP) is a processing model in which each processor in a multiple-processor system executes a single operating system image. The scheduler of the operating system is responsible for scheduling processes on each processor.

This is an efficient processing model when the selected single operating system meets the system requirements. The operating system uses the processing power of multiple processors automatically and is consequently transparent to the end user. Programmers can specify a specific processor to execute a process and handle interrupts with any available processor and designate one processor as the master for system initialization and booting other processors.

1.3.2 Operating system considerations

This section will introduce the operation system in the embedded system and important role of the operating system in the embedded design.

1.3.2.1 Bare-metal system

Bare-metal refers to a software system without an operating system. This software system typically does not need many features (such as networking) that are provided by an operating system. An operating system consumes some small amount of processor throughput and tends to be less deterministic than simple software systems. Some system designs might not allow the overhead and lack of determinism of an operating system [11]. As processing speed has continued to increase for embedded processing, the
overhead of an operating system has become mostly negligible in many system designs. Some designers choose not to use an operating system due to system complexity.

1.3.2.2 Linux operating system

Linux is an open-source operating system used in many embedded designs. It is available from many vendors as a distribution, or it can be built from the open-source repositories. A full-featured operating system takes advantage of the Memory Management Unit (MMU) in the processor, and is consequently regarded as a protected operating system. Linux also provides SMP capabilities to take advantage of multiple processors [11].

1.3.3 Programmable logic

The second important part in the Zynq chip is the Programmable logic (PL). This provides a rich architecture of user configurable capabilities. It includes the components as follows:

- Configurable logic blocks (look-up table, shift register, cascade adder)
- Block RAM (36Kb). That includes dual ports up to 72 bits wide
- Digital Signal Processing DSP48E1slice can compute \((25 \times 18)\) two's complement multiplier and accumulator for high-resolution signal processor.
- XADC converter converts the analog signal to digital signal.
- Integrated interface blocks for PCI Express designs.

1.4 Partial reconfiguration function

In the Zynq-7000, the processors in the PS always boot first, allowing a software centric approach for PL system boot and PL configuration. The PL can be configured as part of the boot process or configured at some point in the future. Additionally, the PL can be completely reconfigured or used with partial, dynamic reconfiguration. Partial reconfiguration allows configuration of a portion of the PL [8]. This enables optional design changes such as updating coefficients or time multiplexing of the PL resources by
swapping in new algorithms as needed. The dynamic loading and unloading can be controlled by a software module. The PL configuration data is referred to as a bit stream.

FPGA technology is able to provide the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. Partial Reconfiguration (PR) provides even more flexibility, allowing the modification of an operating FPGA design by loading a partial configuration file, usually a partial bit stream file [1]. After a full bit file configures the FPGA, partial bit files can be downloaded to modify reconfigurable regions in the FPGA without interrupting the integrity of the applications running on those parts of the device that are not being reconfigured.

Fig. 1.8 shows that function implemented in Reconfigure Block A can be modified by downloading one of several partial bit streams, A1.bit, A2.bit, A3.bit, or A4.bit. With the partial reconfiguration function, the logic in the FPGA design is divided into two different types, reconfigurable logic, and static logic. The light blue of the FPGA block represents static logic and the gray block portion labeled Reconfigure “Block A” stands for reconfigurable logic. The static logic remains functioning and is completely unaffected while loading of a partial bit file. The reconfigurable logic is replaced by the contents of the partial bit stream [12].
There are many reasons for users to select the dynamic reconfiguration of a hardware function on a single FPGA device. It reduces the size of the FPGA device required to implement a given set of functions (in a sequential fashion) as well as power consumption. It provides flexibility in the choices of algorithms or protocols available to an application. Moreover, it accelerates configurable computing in the system. In addition to reducing size, weight, power, and cost, partial reconfiguration enables new types of FPGA designs that are impossible to implement without it.

![Partial reconfiguration flow](image-url)

**Figure 1-9** Partial reconfiguration flow
Implementing a partially reconfigurable FPGA design is similar to implementing multiple non-PR designs that share common logic. Partitions are used to ensure that the common logic between the multiple designs is identical. Fig. 1.9 illustrates this concept. The top box represents the synthesis of HDL source to net lists for each module. The appropriate net lists are implemented in each design to generate the full and partial bit files for that configuration. The static logic from the first implementation is shared among all subsequent design implementations.
2 INTRODUCTION TO THE HARDWARE SYSTEM DESIGN

This section shows the block diagram and connections between the Programmable Logic (PL) and the Processing System (PS) in this research. Fig. 2.1 shows detailed block diagrams of both the PL side and the PS side.

The Programmable Logic (PL) design includes the following IP modules: AXI interconnection, BRAM controller, GPIO connector, GPIO interrupt, MoM algorithm, and Automata State Transitions module. Specifically, dynamic reconfiguration is applied to the Automata State Transitions module (upward diagonal box). The number of state transitions can be changed dynamically by software application. This ability is very useful for automata models that need to change their state transitions dynamically while the system is still operating. The PS is configured with the following I/O peripherals: USB0, Ethernet0, SD0, UART1, I²C0, and GPIO. All I/O peripheral interfaces are configured via MIO pins. All transceiver data is stored in the Block RAM.
2.1 Block RAM design

This block RAM is designed using Verilog code and instantiated in the VHDL top module. The $readmemh command is used to read data from the Block RAM (Fig. 2.2). This data RAM block is dedicated for storing fuzzified input data. The initialization of the RAM is carried out using text files. Every time the Reset signal is asserted, contents of the Block Ram will be initialized in a hex format. The data format for fuzzified inputs is illustrated below.

```
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000111123_678fffffff8753100000
0000000000_0000000000_0000000000_0000000000_0000001236_8fffffff9987310000_0000000000_0000000000_0000000000
0000001111_1124568fffffff99841000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0012368fffffff99875310
```

**RTL code for Ram declaration**

```verilog
/***********************************************/
(* RAM_STYLE="distributed" *)
reg [RAM_WD-1:0] RAM [(2**RAM_ADDR_WD)-1:0];

initial
begin
    $readmemh(TXT_FILE ,RAM);
end
/**********************************************/
```

<table>
<thead>
<tr>
<th>Signals</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr[7:0]</td>
<td>8 bits address to read data from RAM</td>
</tr>
<tr>
<td>We</td>
<td>Write enable, active high when Processor wants to write data to RAM</td>
</tr>
<tr>
<td>Ram_i[31:0]</td>
<td>RAM input. 32 bits data input</td>
</tr>
<tr>
<td>Ram_o[31:0]</td>
<td>RAM input. 32 bits data output</td>
</tr>
</tbody>
</table>

Table 2-1 Block RAM signals
2.2 Mean of maxima algorithm

a. Mean of maxima flow

The calculation of the mean of maxima (MoM) values is performed incrementally while the fuzzified inputs are sampled. Fig 2.3 depicts the algorithm where the number of samples is equal to the number of elements in the universal set. The algorithm simply tests each sample and looks for the low boundary and the high one with the maximum membership value (it is equal to 1). The algorithm then loads the two boundary values to registers. The mean of maxima value is equal to the average of the low and high boundary values.

![Figure 2-3 MoM algorithm](image)

The membership degree value of each element of the digitally represented fuzzy input is compared with the maximum value (0xF for the degree of 1). The maximum hex value depends upon the number of bits used to represent the discrete set of membership values. For example, in this research every discrete element of the fuzzy inputs is represented by a 4-bit code (16 degree of membership levels) with a maximum value of 0xF. If the value of a sample is equal to the maximum value, the position of this element in the universal set will be marked by a one in the register MAX.Comp. Otherwise, it will be set to zero. For instance, if the assigned values for elements ten and eleven are
equal to 0xF, the positions of the tenth and eleventh elements in the MAX_COMP will be set one, respectively. These comparisons are carried out simultaneously. Therefore, all elements of the universal set will be processed in one clock. After the comparisons and the marking process are done, two processes that look for the low boundary and the high boundary, respectively, are executed next. In one these processes, the “for … loop” program structure is used to scan all elements from the first position towards the last position of the universal set to find the low boundary value. Simultaneously, the other process scans all elements from the maximum position towards the first position to find the high boundary value. Because these processes are executed using a “for … loop”, it takes just one clock to conclude the searches. For example in Fig. 2.4, the low boundary value is 30 and the high boundary value is 50. So the mean of maxima for this fuzzy input is \((30+50)/2 = 40\).

![Figure 2-4 Low and high boundary of MoM value](image)

VHDL program to generate-N comparators:

```vhdl
/*******************************************************************************/
GEN_LARGECOMP: for i in 0 to N_COM-1 Generate
  Gen: compare port map ('
da_i => da_i((i+1)*N_BIT-1 down to i*N_BIT),
da_o => larg_da_o(i));
end generate;
/*******************************************************************************/
```
VHDL program to look the high boundary value:

```vhdl
function Anyones_upper_limit(v : in std_logic_vector(ELE_NUM-1 down to 0)) return integer is
  variable i : integer range 0 to ELE_NUM-1 := 0;
begin
  for i in ELE_NUM-1 down to 0 loop
    if v(i) = '1' then
      return(i+1);
    end if;
  end loop;
  return 1;
end function;
```

VHDL program to look for the low boundary value:

```vhdl
function Anyones_lower_limit(v : in std_logic_vector(ELE_NUM-1 down to 0)) return integer is
  variable i : integer range 0 to ELE_NUM-1 := 0;
begin
  for i in 0 to ELE_NUM-1 loop
    if v(i) = '1' then
      return(i+1);
    end if;
  end loop;
  return 1;
end function;
```

b. Mean of maxima block diagram

![Figure 2-5 MoM block diagram](image)

Figure 2-5 MoM block diagram
<table>
<thead>
<tr>
<th>Signals</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Ctrl_en</em></td>
<td>Control enable signal, active high in 1 clock. It is asserted when new data coming</td>
</tr>
<tr>
<td><em>Data_i(99:0)</em></td>
<td>Data input from Max_comp register. It is marked ‘1’ when sample value equal to Maximum value (0xF) otherwise is ‘0’</td>
</tr>
<tr>
<td><em>Da_valid</em></td>
<td>Data output valid, active high. This is asserted when new mean of maxima value output is valid</td>
</tr>
<tr>
<td><em>MoM_o(6:0)</em></td>
<td>mean of maxima value.</td>
</tr>
</tbody>
</table>

Table 2-2 Mean of maxima signals

### 2.3 Automata state transitions

#### a. Block diagram

Fig. 2.6 shows the interface of the Automata State Transitions module. The results of the mean of maxima (MoM) computations are grouped together with the current state information as inputs to the state transitions module. The *ctrl_en* signal is asserted when the validation signals of the MoM are asserted. True or false values of the next state transition functions are computed. One of the state transition functions should be evaluated as true. The next state value will be determined accordingly.

![Automata State Transitions](image_url)

Figure 2-6 Automata state transitions module
b. Flow chart

![Flow chart diagram]

Figure 2-7 State transition flowchart

VHDL program to compute automata state transitions

/*************************************************************/

NXT_STATE: process (prs_sta,nxt_sta,acc_i,time_i)
begin -- process NXT_STA
    nxt_sta <= prs_sta;
    case prs_sta is
        when S1 =>
            if (unsigned(acc_i) < A_BA) or (unsigned(time_i) > T_BA) then
                nxt_sta <= S2;
            elsif (unsigned(acc_i) < A_AA) or (unsigned(time_i) > T_AA) then
                nxt_sta <= S3;
            elsif (unsigned(acc_i) <= A_E) and (unsigned(time_i) >= T_E) then
                nxt_sta <= S4;
            end if;
...........
end process

/*************************************************************/

26
2.4 Simulation waveforms

![Simulation waveforms using ISIM](image)

Figure 2-8 Automata state transition waveforms using ISIM

For the sake of easier monitoring, the simulated signals are grouped into five groups that are named such as Fuzzified input, MoM timing, MoM accuracy, and State Transition, as depicted in Fig. 2.8. Every time when the user hits a push button an external interrupt is received by the system. At this time, new fuzzified data are retrieved from the Block RAM. Having received the data, the mean of maxima of output data is computed and those values are passed onto the state transitions module. Depending upon the MoM values and the results of the evaluations of the state transition functions, in the Automata State Transitions module (either 12 states or 15 states) the value of the next state will be determined and registered. A new cycle will be processed when it the system receives another interrupt from the user.
2.5 ISE summary report

According to the Summary Report (Fig 2.9), there are no errors, neither any warnings for this design by the Synthesis and Implementation checking. The RTL code accounts for the relatively low utilizations of the available hardware resources. All timing and logic problems were debugged and removed by using ISIM test-bench simulations and ChipScope. All specified behaviors of the RTL design worked correctly both in the ISIM simulation runs and in the board-level demonstrations.
<table>
<thead>
<tr>
<th>Report Name</th>
<th>Status</th>
<th>Generated Date</th>
<th>Errors</th>
<th>Warnings</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis Report</td>
<td>Current</td>
<td>Fri Nov 8 15:16:54 2013</td>
<td>0</td>
<td>0</td>
<td>1 Failure (1 nets)</td>
</tr>
<tr>
<td>Translation Report</td>
<td>Current</td>
<td>Fri Nov 8 15:16:57 2013</td>
<td>0</td>
<td>0</td>
<td>1 Failure (1 nets)</td>
</tr>
<tr>
<td>Place and Route Report</td>
<td>Current</td>
<td>Fri Nov 8 15:18:41 2013</td>
<td>0</td>
<td>0</td>
<td>1 Failure (1 nets)</td>
</tr>
<tr>
<td>Power Report</td>
<td>Current</td>
<td>Fri Nov 8 15:18:42 2013</td>
<td>0</td>
<td>0</td>
<td>1 Failure (1 nets)</td>
</tr>
</tbody>
</table>

### Performance Summary

<table>
<thead>
<tr>
<th>Final Timing Score</th>
<th>0 (Setup: 0; Hold: 0; Component Switching Latency: 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-Place and Route Simulation Model Report</td>
<td>Out of Date</td>
</tr>
<tr>
<td>Synthesis Report</td>
<td>Fri Nov 8 15:16:54 2013</td>
</tr>
<tr>
<td>Place and Route Simulation Report</td>
<td>Fri Nov 8 15:18:41 2013</td>
</tr>
<tr>
<td>Power Report</td>
<td>Fri Nov 8 15:18:42 2013</td>
</tr>
</tbody>
</table>

Figure 2-9 ISE summary report
3 PARTIAL RECONFIGURATION AND SOFTWARE DESIGN

This section provides for more details about the dynamic partial reconfiguration design method. In this project, the Automata State Transition module is identified as a good candidate for partial reconfiguration (PR) because the state transition graph implemented is independent application and will be changed. This section also covers topics with respect to the operating system, the processing subsystem (PS) configuration and the programmable reconfiguration design with the FPGA Zynq-7000 board.

3.1 Partial reconfiguration design

The following terminology is used for design. The reconfigurable partition (RP) refers to the physical location on the FPGA that is selected for partial reconfiguration while the remainder of the design and logic are referred to as static logic. For this project, the Automata State Transitions module is considered as dynamic logic. It is defined by its hardware interfaces and ports and they are mapped onto the reconfigurable partition. A specific state transition graph can be implemented in this partition and it is called as a reconfigurable module (RM). A reconfigurable module, in conjunction with the static logic, is named as a configuration. The two dynamic configurations in this design are called 12 states and 15 states configurations, respectively. A configuration defines a complete FPGA design and produces a full bit stream that includes both static and reconfigurable logic while the partial bit stream only includes the partial configuration.
logic. All full bit streams and partial bit streams are stored in the DDR memory during the boot process. These bit streams can be reconfigured into the PL by software (Fig. 3.1).

### 3.1.1 Automata transition state operation

To understand the implications of our design choices, we first look at the operation of the Automata State Transitions module. This module has an interface with the MoM module and the current state to decide which state should be the next one. First, this module needs to be configured with the 12 states configuration and the initial state is State 1. Next, the fuzzified input data of the first trial is read from the Block RAM memory by hitting the interrupt push button. This way the system can process one by one each trial data. Each trial includes all elements of the universal set. In this example design, the universal sets of the two fuzzified inputs (time and accuracy) include 100 elements.

### 3.1.2 Hardware interface resources

This partial reconfiguration design for the Automata State Transition designs module has the following hardware interfaces and connections to the static segment of the system.

- 32-bit AXI4-Lite interface connected to the GP0 Master Interface
- Interrupt signal connected to PS-GIC
- GPIO interface controller
- Clock signal connected to the same clock domain as AXI4-Lite (100 MHz in this design).

When choosing an IP module for partial reconfiguration, the designer has to make sure that the hardware interfaces and ports of all reconfigurable modules are identical with respect to the static portion of the system. In the partial reconfiguration design, ports are inserted at the boundary of the reconfigurable partition and the static partition of the design. Therefore, all reconfigurable modules need to share the same ports at the same locations with respect to the static logic.
3.1.3 File structure

a. RTL program

Code

Static ................................................................. Static folder
Top.v .......................................................... Top static file
Compare.v .................................................. 4-bit compare
Largecompare.v .............................................. 400-bit compare
Fuzzy_ctrl.v ........................................ Control fuzzy input data
MoM.v ...................................................... Mean of Maxima
RAM.v ............................................ Distribution RAM
Edge_dec.v ........................................ Falling or Rising edge detection
Debcrr_free.v ........................................ De-bouncer push buttons
12 states ..................................................... Partial Reconfiguration folder
   12_state_trans.v ........................................ 12 states Partial Reconfiguration file
15_states .................................................... 15 states Partial Reconfiguration folder
   15_states_trans.v ........................................ 15 states Partial Reconfiguration file
Pins.ucf ........................................................ UCF file.

b. Software code

SDK_export .................. SDK workstation
   Standalone_bsp_0 ........... All drivers, libraries for ARM core
   Hw_system_platform ...... All hardware exported from Plan Ahead and XPS design.
   Ps_7_init.c ............... Initial ARM core
   System.bit .............. All system bit file
Partial_Reconfiguration ..... Software design
   src ............................ Contains all C source files (main_pr.c, platform.c).
   Debug ......................... Contains all debug source files for SDK debugging.
3.1.4 PS configuration design

The PS is configured with the following I/O peripherals enabled (see Fig. 3.2) USB0, Ethernet0, SD0, UART1 and GPIO. All I/O peripheral interfaces are configured for MIO. A PS internal clock generator provides the 100 MHz clock to the PL. The PS connects to PL via the AXI bus interface and data are written to PL via this AXI connection by the software program. An external control signal from PL will be connected to PS via the global interrupt controller (GIC) and the interrupt service routine is programmed onto the PS to handle this interrupt. In this design, we use the BSP (bare-metal) library and drivers to write the embedded software. In addition, the ChipScope monitor IP is added to monitor the AXI interface where the data transfers between PS and PL take place.
3.1.5 Partial reconfiguration design flow

Fig. 3.3 shows the general design flow for partial reconfiguration. The steps below provide for more details about this flow.

Firstly, the system hardware design is entered as a Plan Ahead/XPS project, which has the 12 States IP already instantiated. The 12 States module design is
synthesized to generate the corresponding net lists and constraint files for the overall system and the other IP cores. Secondly, after finishing the first synthesis, the 12 States module is replaced with the 15 States module and the new design is re-synthesized. From this step, we only extract the net list and constraint file for the 15 States module. All other net lists and constraint files are taken from the previous step. After all the necessary IPs have been synthesized, the individual steps of the Plan Ahead PR design flow [1] should be carried out as described below.

Firstly, we create a Plan Ahead PR project targeting the ZC702 evaluation platform and import the net lists and constraint files generated in the previous steps, except for the 12 States and 15 States modules’ net list and constraint files. This represents the static logic of the design. Secondly, after loading the synthesized design the Automata State Transitions module is treated as a black box because there is no net list associated with it. Define the Automata State Transitions module as a Reconfigurable Partition (RP). Partitions ensure that the logic and routing common to each of the multiple designs are identical. Next, two Reconfigurable Modules (RM) are created by adding the corresponding net list/constraint files for the 12 and 15 States modules (Fig. 3.4). Constraints that only apply to specific Reconfigurable Modules (RMs) must be copied to the module level and should be provided alongside the corresponding net lists. Constraints applied to the static logic and any constraints that are shared across all RMs should be included in the top-level constraint file.

![figure 3-4 reconfigurable modules](image)

Figure 3-4 Reconfigurable modules
After that, create a floor plan for the Reconfigurable Partition by setting the physical size of the partition and the types of resources desired. Xilinx FPGAs support reconfiguration of Flip-Flops, distributed RAM, Multiplexers, the BRAM and the DSP blocks, plus all the associated routing resources. The designer needs to design the floor plan for the reconfigurable module such that it can accommodate all the resources required by the reconfigurable modules. As a rule of thumb, approximately twenty percent overhead should be accounted for routing resources (Fig. 3.5; Fig. 3.6).

The next step is to set the configuration for the reconfigurable design. The first configuration to be chosen for implementation should be the most challenging one. If all reconfigurable modules in the subsequent configurations are smaller or slower, it will be easier to meet their demands. We first implement the 12 States configuration. The Plan Ahead tool ensures that the resources used to construct the reconfigurable modules are completely contained within the defined physical region of the RP and that no interference with the static portion of the design occurs. After successful implementation, promote the 12 States configuration such that the implementation results of the static portion of the design can be reused by the 15 States configuration (Fig. 3.7).
Having successfully implemented the 12 States configuration, next we implement the 15 States configuration. Make sure to import the static logic from the 12 States configuration that was generated in the previous step. Then we run the PR Verify Configuration Utility to validate consistency between the implementations of the 12 States and 15 States configurations.
Finally, we need to generate the full and partial bit streams for the both configurations. We will use the full bit stream of the 12 States configuration as the default start-up configuration when booting the Zynq device. These bit streams are used for the dynamic configuration of the FPGA board when we run the software and hardware application.

### 3.1.6 Device configuration and boot flow

To understand the configuration flow and boot flow of the Zedboard we should investigate the function of the AXI-PCAP interface. The main function of the AXI-PCAP Bridge is to convert the 32-bit AXI formatted data to the 32-bit PCAP protocol and vice versa. There are transmitting and receiving FIFO data buffers between the AXI and the PCAP interface. A Direct Memory Access (DMA) engine moves the data between the FIFOs and the memory devices, e.g., the DDR memory, or one of the peripheral memories. The PCAP interface is clocked at 100 MHz for PL configuration. In order to transfer data across the PCAP interface we use the DevC driver function. This driver will take care of the setup of the correct PCAP mode and the initiation of the DMA transfer [9]. 

*Fig. 3.8* shows that after power-on reset the Boot ROM determines the use of either the external memory interface or the boot mode SD flash memory. The First Stage Boot Loader (FSBL) will be loaded into the on-chip RAM by using the Device Configuration Interface DMA (DevC’s DMA). After that, the Boot ROM shuts down and releases the CPU control to the FSBL to configure the programmable logic PL with the full 12 States bit stream via the Processor Configuration Access Port (PCAP) (*Fig. 3.8*). The device is now fully configured and operational. With the standalone driver, the FSBL loads and releases control to the standalone user application.
After loading the full bit stream and if the system operates normally, the user application can load the partial bit stream into the DDR memory upon start-up. This step is to increase the configuration throughput over the PCAP interface and, hence, to reduce the configuration time and take advantage of caching. At this point, the user application can load the partial bit streams at any time to modify the pre-defined PL regions while the rest of the FPGA remains fully active and uninterrupted. This is done by transferring the partial 12 States bit stream or the partial 15 States bit stream from the DDR to the PL via PCAP. The loading of a partial bit stream into the PL does not require knowledge of the physical location of the reconfigurable module, since the configuration address information is included in the partial bit stream file.
3.1.7 Plan Ahead summary report

Figure 3-9 Plan Ahead summary report

Figure 3-10 Floor plan report

In this summary report, we have no errors and no warnings in the Partial Reconfiguration Plan Ahead (Fig. 3.9). The partial reconfiguration module works perfectly when the full bit stream or the partial bit stream is downloaded to the FPGA. The amount of the required hardware resource takes a very small percentage of the available FPGA resources. Moreover, the physical floor plan and timing constraint in the PR projects have no warnings (Fig. 3.10).
3.2 Embedded software design flow

The most advantageous feature of the Zynq-board is the dual ARM-Core processors that users can program such that the embedded code controls the behavior of the programmable hardware. The software flow that is used to configure hardware dynamically will be discussed in this section in detail.

3.2.1 General embedded design flow

![Diagram of general software design flow]

Figure 3-11 General software design flow

Fig. 3.11 introduces the general flow to design the embedded software for the Zedboard by the Plan Ahead, the Xilinx Platform Studio (XPS), and the Software Development Kit (SDK) tools. The Plan Ahead software provides a central project for design entry in RTL, synthesis and verification. In Plan Ahead, we create the project and the top module design to connect the system processor and IP instances designed by other users. In addition, Plan Ahead offers integration with XPS for embedded processor design peripheral IPs and peripheral interfaces. In XPS, the designer can add the ARM
processor embedded system and other available IPs, the AXI interface, high speed and low speed buses that provide connections between the PS and the PL on the FPGA chip. All hardware configurations, peripheral IPs and bus connections are created in the XPS software. After the designed hardware is configured and stable, the user can return to the Plan Ahead tool and export all hardware resources to the SDK software to complete the embedded processor software design. When working with the SDK software tool, there are two possible ways for the designer to write the embedded software: either by using the Board Support Package (BSP) bare-metal or by using Linux OS drivers and a kernel to access to the hardware. After finishing the programming and compiling the software application, both the hardware bit stream and the software .elf files will be downloaded to the Zynq board.

![Figure 3-12 Software interface flow](image)

*Fig. 3.12* shows the software application flow for this design. Firstly, when power is turned on, the embedded system is booted and initialized during the FSBL boot
process. Then the fuzzified data will be transferred to the Block RAM by software. In addition, the full bit streams and partial bit streams are also transferred to the internal DDR3 by software. The purpose of this behavior is to improve the speed of execution of the configuration process. After the data transfer and bit stream processes are successfully executed, the user can configure the partial reconfiguration modules by selecting the software options from the UART interface that is connected to Tera-term. The software interface is displayed in the Fig. 3.13. After the hardware configuration process for the reconfigurable modules is finished, the new fuzzified data will be read from the Block RAM by hitting the push button. The new state of the fuzzy automata will be updated if the corresponding state transition condition is satisfied. The cycle is repeated by selecting the reconfiguration options.

3.2.2 Software interface

![Software interface figure]

Figure 3-13 User interface

Fig. 3.13 displays the software interface of the dynamic partial reconfiguration. There are five options for the users to configure the hardware. The users can type their
selection from the keyboard. This port will interface with the UART port in the ARM processor system. With option1, option2 will run without partial reconfiguration. When option1 and option2 are selected, the 12 States and 15 States full bit streams will be configured for the Automata State Transitions module, respectively. Option3 and option4 are used for partial reconfiguration. The Automata State Transitions module will work with 12 States and 15 States partial bit streams by selecting option3 or option4, respectively. The user can stop the program by selecting option5.

The function below is used to initialize the GPIO ports and the devices. It returns the XST_SUCCESS if the GIPOs are ready to work, otherwise it returns XST_FAIL

```c
/**********************************************************************************
 * Initialize the GPIO driver
 */
Status = XGpio_Initialize(&Gpio, GPIO_EXAMPLE_DEVICE_ID);
if (Status != XST_SUCCESS) {
 return XST_FAILURE;
}
/**********************************************************************************

The function below is used to initialize the SD card and to transfer the partial bit files from the SD to the DDR memory. If there are any errors, the function returns the message ERROR

```c
/**********************************************************************************
 // Initialize SD controller and transfer partials to DDR
 SD_Init();
 SD_TransferPartial("pr_12states.bin", PARTIAL_SOBEL_ADDR, ARTIAL_BITFILE_LEN);
 SD_TransferPartial("pr_15states.bin", PARTIAL_SEPIA_ADDR, PARTIAL_BITFILE_LEN);
/**********************************************************************************

This function is used to transfer a bit stream from the DDR to the PL via DevCfg/PCAP. After the loading of the transfer bit stream to DDR, this function is used to read the data from the DDR and to transfer them to the PL for reconfiguration.

```c
/**********************************************************************************
 void Configure_ParRecfg(XDcfg *XDcfg, u32 PartialAddress, u32 PARTIAL_BIT_LEN)
{
    // Transfer Bitfile using DEVCFG/PCAP
    int Status = XDcfg_TransferBitfile(XDcfg, PartialAddress, (PARTIAL_BIT_LEN >> 2));
    if (Status != XST_SUCCESS) {
        xil_printf("ERROR : FPGA configuration failed!

        exit(EXIT_FAILURE);
    } 
} 
/**********************************************************************************
4 MATLAB AND ISIM SIMULATIONS

The Intelligent Decision Support System (IDSS) for Eye-Hand Coordination Assessment is used as a target system for this Thesis project. The goal objective for the IDSS was to develop an automated assessment and training procedure for children with eye-hand coordination problems. In this section, we will not be concerned much about the strategy to research eye-hand coordination. We just use its database and the state transition conditions table for our application. The main purpose of using eye-hand coordination application is to implement this fuzzy automata model on the FPGA board and run it in real-time. In this Thesis, the 12 States transition graph has been developed for children of age group 5 from the actual eye-hand coordination research. While the 15 States transition graph is extended from the 12 States graph, it does not reflect the results of any occupational therapy research. That state transition graph is used as an example only for this research. This example application runs in real-time on Zedboard FPGA board with dynamic partial reconfiguration. The results are verified by Matlab simulations.

4.1 Matlab design

In this project, Matlab simulations are also carried out to double-check the results delivered by the hardware accelerator. This section will discuss the block diagram and the flow chart of the MatLab program design and simulations.

4.1.1 Block diagram

Fig. 4.1 shows the block diagram for Matlab model. This Matlab program simulates the behavior of the FPGA hardware in order to double check the results. The fuzzified input patterns are the same as the fuzzified inputs for the hardware module. For the Matlab program module, the fuzzified time and accuracy input vectors are taken from the time.dat and accuracy.dat files, respectively. Then these data will be transferred to create the accuracy and time matrices. The sizes of the matrices depend on the number of elements in the universal set and the number of bits used to represent the discrete set of
membership values for each element. For example, if the number of elements in the universal set is 100 and each element is represented by 16 levels (4 bits), then the fuzzified input vectors will be of 400 bits long for each pattern. After creating the matrix for each input vector, this summary data will be moved to the mean of maxima model. The outputs of the MoM module along with the present state information and the state transition conditions for each state are used to devise the next state of the automata. Details of the Matlab modules will follow in the next section.

Figure 4-1 Mat lab design and block diagram

4.1.2 Input data format

In this project, we have two files of data that are named as time.dat and acc.dat. Each line of the file represents a fuzzified vector input in a particular present state.

<table>
<thead>
<tr>
<th>Time</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>0.1</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>0.6</td>
<td>0.3</td>
</tr>
<tr>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

This format implies that starting from the universal set element 34 the membership values for fuzzy input Accuracy will be 0.1, 0.3 etc. The membership values for the rest of the elements in the universal set are termed as 0s. The same format applies for the fuzzy input Time Taken.
4.1.3  Create matrix function

The file name is *create_matrix.m*. The main function of this module is to read the fuzzified data from the fuzzified input files (*acc.dat; time.dat*) and then create the matrices that will hold all data from the input files. The fuzzified input data will be then transferred to the mean of maxima module for computing.

Matlab code for the Create matrix function

```matlab
function [new_ma] = create_matrix(input_ma)
% Get row and column from input matrix;
[row, col] = size(input_ma);

% Create the Fuzzy inputs matrix from the data files
new_ma = zeros(row, 101);
for i = drange(1:row)
    index = input_ma(i,1) + 1;
    for j = drange(2:col)
        new_ma(i, j+index-2) = input_ma(i,j);
    end
end
```

4.1.4  Mean of maxima module

This module is to find the mean of maxima value for the fuzzified inputs of every trial. The inputs of this module are taken from the data matrices while the output of the MoM is passed onto the Automata State Transitions module. The calculation of the MoM values is done incrementally while the fuzzy inputs are being sampled. *Fig. 4.2* shows the algorithm where the number of samples is equal to the number of elements in the universal set. The algorithm simply tests each sample for a maximum value and records this value. Each time the maximum value is received at a particular input, a counter is incremented. In addition, the corresponding sample count is added to a running sum. When a new maximum value is sampled, this new value will be recorded as the new maximum number and the running sum will be reset back to the corresponding sample count of the new value. This algorithm is different from the algorithm applied in the RTL code but it yields the same MoM value.
Figure 4-2 Mean of maxima flowchart

```plaintext
function mom_o = MOM(x)
[row, col] = size(x);
mom_o = zeros(1, row);
for i = drange(1:row)
  max_x = 0;
  total = 0;
  cnt = 0;
  boolean = zeros(1, 4);
  for j = drange(1:col)
    if x(i, j) > max_x
      max_x = x(i, j);
      total = j;
      cnt = 1;
    elseif x(i, j) == max_x
      cnt = cnt + 1;
      total = total + j;
    end;
  end;
end;
MOM = total/cnt;
```

/******************** Mean of maxima flowchart */
4.1.5 State transitions

The results of the MoM algorithm along with the current state information become the inputs of the Automata State Transitions module. If the input vectors match with the conditions listed for a state transition then the designated next state becomes the present state. If no match is found the present state is retained. The 12 States and the 15 States transition conditions are given in Tables 4-1 and 4-2.

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Transition Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>Time A and/or Accuracy A</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Time AA and/or Accuracy AA</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>Time X and Accuracy X</td>
</tr>
<tr>
<td>9,10,11</td>
<td>12</td>
<td>Stop</td>
</tr>
<tr>
<td>9,10,11</td>
<td>12</td>
<td>BA Below Average</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A Average</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AA Above Average</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E Excellent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X Don’t care</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/ Or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A and/or B A and B, A or B</td>
</tr>
</tbody>
</table>

Table 4-1 12 States transitions conditions
<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Transition Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>Time A and/or Accuracy A</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>Time AA and/or Accuracy AA</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>Time BA and/or Accuracy BA</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>Time A/AA and/or Accuracy A/AA</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>Time E and Accuracy E</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>Stop</td>
</tr>
</tbody>
</table>

BA = Below Average
A = Average
AA = Above Average
E = Excellent
X = Don’t care
/ = Or
A and/or B = A and B, A or B

Table 4-2 15 States transitions conditions
Figure 4-3 12 states transition graph
Figure 4-4 15 states transition graph
function state_o = state_transition(x,y)
    [row,col] = size(x);
    state_o = zeros(1,col);
    state_o(1) = 1;
    prs_state = 1;
    for i = drange(1,col+1)
        state_o(i) = prs_state;
        switch prs_state
            case 1
                if (x(i) < 40) | (y(i) > 75)
                    nxt_state = 2;
                elseif (x(i) < 70) | (y(i) > 25)
                    nxt_state = 3;
                elseif (x(i) <= 100) & (y(i) >= 0)
                    nxt_state = 4;
            end
            case 2
                if (x(i) < 40) | (y(i) > 75)
                    nxt_state = 1;
                elseif (x(i) < 70) | (y(i) > 25)
                    nxt_state = 3;
                elseif (x(i) <= 100) & (y(i) >= 0)
                    nxt_state = 4;
            end
            case 8
                nxt_state = 7;
            otherwise
                nxt_state = 12;
            end
        end
    end
    prs_state = nxt_state;
end
/***************************************************************************/

4.2 Evaluation of the results

ISIM simulations and Matlab simulations are used to verify the output results of
the hardware accelerator design. In this section, we will provide more details about the
ISIM simulation waveforms and the Matlab simulation results.

4.2.1 Definitions and input data

The universal set is the set that consist of all the elements of interest for a
particular application. It is the mother of all fuzzy sets; any set that is not a universal set
is a subset [13]. In this project, the universal set has 100 elements and each element is
represented by 4 bits (16 membership levels), therefore, the total number of bits for one fuzzified input is 400. The universal sets used in both the Matlab model and the hardware implementation are depicted in **Fig. 4.5**. The sub-intervals used for the fuzzy inputs Accuracy (given in percent) and Time (given in seconds) are normalized to just one uniform set to simplify the hardware implementation. The accuracy input is defined with four sub-intervals ranging from below average (BA) to excellent (E) while the time is also defined with four sub-intervals ranging from excellent (E) to below average (BA). In other word, the accuracy input smaller than 40% is defined as below average (BA), between 40% and 55% is called average (A), if the accuracy is bigger than 55% and smaller than 70%, it is defined as above average (AA) while the accuracy input bigger than 70% is named excellent (E). Similarly, for the time input, we have time input smaller than 40% is defined as Excellent (E), between 40% and 55% is called above average (AA), the time larger than 55% and smaller than 70% is defined as average (A) while time input larger than 70% is named below average (BA).

![Figure 4-5 Universal set for time and accuracy inputs](image)

**a. Fuzzified Accuracy input format**

```
0000000000_0000000000_0000000000_0000000000_0000001236_68ff873f998_710000_0000000000_0000000000_0000000000
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
```

```
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000_0000000000
```
b. Fuzzified time input format

The 12 States transition graph implemented in this design is shown in Fig. 4.8. This state transition graph has been developed for children of age 5. It is based upon experimental knowledge provided by occupational therapy experts. The state transition graph depicts a path to improve eye-hand coordination skills. Starting at state1 and reaching to state12 (or state1 to state15) 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.
4.2.2 Mat lab simulation

a. Test case 1

Figure 4-6 Defuzzified time input in second

Figure 4-7 Defuzzified accuracy input in %
Figure 4-8 12 states transitions

Figure 4-9 15 states transitions
b. Test case 2

Figure 4-10 Defuzzified time in second

Figure 4-11 Defuzzified accuracy in percent
In this simulation, we have 20 trials with different fuzzified inputs for both the 12 States and the 15 States transition graphs. Two test cases were run for this verification. Defuzzified values are shown in Fig. 4-12, Fig. 4-13 and Fig. 4-14, Fig. 4-15 for test case1 and test case2, respectively. The visited states in the 12 States and the 15-States graphs
with Matlab and real-time FPGA simulation are shown in Fig. 4.8, Fig. 4.9, Fig. 4.12 and Fig. 4.13 for test case1 and test case2, respectively.

4.2.3 ISIM simulation

<table>
<thead>
<tr>
<th>Group signals</th>
<th>Signals</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzified input</td>
<td>ram_time_o (399:0)</td>
<td>Fuzzified Time input</td>
</tr>
<tr>
<td></td>
<td>ram_acc_o (399:0)</td>
<td>Fuzzified Accuracy input</td>
</tr>
<tr>
<td>MoM time</td>
<td>comp_o(99:0)</td>
<td>Compare register. This register will mark the time samples that have value = 0xf (4bits)</td>
</tr>
<tr>
<td></td>
<td>lower_limit</td>
<td>The low boundary of mean of maxima</td>
</tr>
<tr>
<td></td>
<td>upper_limit</td>
<td>The high boundary of mean of maxima</td>
</tr>
<tr>
<td></td>
<td>uom_tim_o(6:0)</td>
<td>The value of time MoM</td>
</tr>
<tr>
<td>MoM accuracy</td>
<td>comp_o(99:0)</td>
<td>Compare register. This register will mark the accuracy samples that have value = 0xf (4bits)</td>
</tr>
<tr>
<td></td>
<td>lower_limit</td>
<td>The low boundary of Mean of Maxima</td>
</tr>
<tr>
<td></td>
<td>upper_limit</td>
<td>The high boundary of Mean of maxima</td>
</tr>
<tr>
<td></td>
<td>mom_acc_o(6:0)</td>
<td>The value of accuracy MoM</td>
</tr>
<tr>
<td>State transition</td>
<td>ri_ctrl</td>
<td>Rising edge of control signal</td>
</tr>
<tr>
<td></td>
<td>nxt_sta_o(3:0)</td>
<td>Next state output</td>
</tr>
</tbody>
</table>

Table 4-3 Top signal explanation
4.2.4 ChipScope waveform

This design is not only tested in a test-bench environment with the ISIM simulator but also run on the ChipScope tool. The ChipScope waveform is shown in Fig. 4.15, new data are captured every rising edge of the external interrupt trigger signal (ri_ctrl). All ISIM simulations and the FPGA implementation work perfectly well. The behavior of the hardware accelerator matches the desired specifications as well as the Matlab simulations.
5 CONCLUSIONS AND FURTHER INVESTIGATION

In this Thesis, the design successfully runs both on the FPGA-based hardware implementation and with Matlab. By using the Partial Reconfiguration function on the Zedboard and developing the embedded software application, it is proven that this design can be applied for implementing dynamically reconfigurable state transitions for fuzzy automata. The dynamic partial reconfiguration capability is a very attractive property to implement virtual fuzzy automata for supervisory controllers of complex systems. This virtual fuzzy automata architecture allows the supervisory controller to change the fuzzy automaton under real-time conditions in order to model a particular state cluster, as needed. Another important result of this project that this design is controlled by embedded software that runs on the ARM Core. Hence, it provides a great deal of flexibility for users to develop more functions in the future by improving and integrating more software tasks. The results of this design are independently verified by Matlab simulations and in a Testbench.

For future work, the development of reconfigurable inference engines and hardware accelerators for creating the knowledge base unit are considered. In addition, the fuzzified data will be generated and received in real-time via Giga bit Ethernet from remote devices. It will make the system is more flexible and it more powerful. With respect to the embedded software development the Linux operating system and Linux, drivers will also be considered for future research.
REFERENCES


Xilinx, Partial Reconfiguration Tutorial PlanAhead Design Tool, Xilinx, Jan 2012.


GRANTNER, JANOS L.; GEORGE A. FODOR; MAREK J. PATYRA, APPLICATION OF THE FUZZY STATE FUZZY OUTPUT FINITE STATE MACHINE TO THE PROBLEM OF RECOVERY FROM VIOLATIONS OF ONTOLOGICAL ASSUMPTIONS.

Xilinx, Zedboard Schematic, Xilinx, Jan 2013.


