Wright State University

CORE Scholar

[Browse all Theses and Dissertations](#)

[Theses and Dissertations](#)

2006

Implementation of a Multi-Layered Fuzzy Controller on an FPGA

Ajit Paal Singh Khatra
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the [Electrical and Computer Engineering Commons](#)

Repository Citation

Khatra, Ajit Paal Singh, "Implementation of a Multi-Layered Fuzzy Controller on an FPGA" (2006). *Browse all Theses and Dissertations*. 35.

https://corescholar.libraries.wright.edu/etd_all/35

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.

IMPLEMENTATION OF A MULTI-LAYERED FUZZY CONTROLLER ON AN FPGA

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering

By

Ajit P Khatra

B.Tech., Punjab Technical University, 2002

2006

Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

July 3, 2006

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Ajit P Khatra ENTITLED Implementation of a Multi-Layered Fuzzy Controller on an FPGA BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering.

Kuldip S. Rattan, Ph.D.
Thesis Director

Fred D. Garber, Ph.D.
Department Chair

Committee on
Final Examination

Kuldip S. Rattan, Ph.D.

John M. Emmert, Ph.D.

Pradeep Misra, Ph.D.

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

ABSTRACT

Khatra, Ajit Paal Singh. M.S. Egr., Department of Electrical Engineering, Wright State University, 2006. Implementation of a Multi-layered Fuzzy Controller on an FPGA.

The uses of Autonomous Mobile robots range from the teleoperated Sojourner on the Mars Pathfinder mission to cleaning robots in the Paris Metro. They have overtaken tasks which are too difficult or dangerous to perform or require the repetition of the same procedure over and over again and can become dull for a human to perform. The objective of this thesis is to develop and implement a multilayered fuzzy controller on an FPGA for the navigational control of an autonomous mobile robot (AMR). The navigational algorithm assists the robot in navigating through complex environments of passages and hallways. The FPGA implementation of the navigational algorithm employs a layered fuzzy control strategy where a control model is developed using three elementary layers of control: Centering Layer, Wall Hugging Layer, and Obstacle Avoidance Layer. The centering layer is used to control the robot motion through a passage with both walls within the range of robot side sensors. In such a scenario, this layer directs the robot to the center of the passage. If the side sensors for the robot detect wall only on one side, wall hugging layer is activated and the robot moves parallel to the side wall. These two layers are developed using a proportional-plus-derivative fuzzy controller and share the fuzzy sets and the rule base. The obstacle avoidance layer prevents the collision of the robot with objects and is activated if the front sensor detects an obstacle within a preset distance. The advantages of FPGA implementation over a microcontroller implementation include faster processing power, compact design and elimination of inter-module communication on different platforms results in improved response of the robot in dynamically changing environment.

TABLE OF CONTENTS

1. <u>INTRODUCTION</u>	1
1.1. BACKGROUND	3
1.2. PROBLEM DESCRIPTION	4
1.3. OUTLINE OF THESIS	7
2. <u>INTRODUCTION TO FUZZY LOGIC</u>	8
2.1. BASIC ELEMENTS OF FUZZY LOGIC	10
2.1.1. FUZZY SETS	10
2.1.2. FUZZY SET TERMINOLOGY	12
2.1.3. FUZZY SET OPERATORS	13
2.1.4. FUZZY PARTITIONING	16
2.1.5. LINGUISTIC VARIABLES	17
2.1.6. FUZZY CONTROL RULES	18
2.2. FUZZY LOGIC CONTROLLER	19
2.2.1. FUZZIFICATION	20
2.2.2. INFERENCE EVALUATION	21
2.2.3. DEFUZZIFICATION	23
2.3. SUMMARY	24

3. <u>HARDWARE DESCRIPTION</u>	25
3.1. SENSOR MODULE	26
3.2. MOTOR DRIVER MODULE	30
3.3. SUPERVISOR MODULE	34
3.3.1. FIELD PROGRAMMABLE GATE ARRAY	35
3.3.1.1.CONFIGURABLE LOGIC BLOCK	37
3.3.1.2.PROGRAMMABLE INPUT/OUTPUT BLOCK	38
3.3.1.3.PROGRAMMABLE INTERCONNECT	39
3.3.2. VIRTEX-II PRO LC DEVELOPMENT BOARD	40
3.4. INTERFACE CIRCUITS	43
3.4.1. 5V TO 3.3V CONVERTER	43
3.4.2. 3.3V TO 5V CONVERTER	45
3.5. SUMMARY	47
4. <u>NAVIGATIONAL CONTROL ALGORITHM</u>	48
4.1. CENTERING LAYER	50
4.1.1. INPUT FUZZY SETS	51
4.1.2. RULE BASE	53
4.1.3. OUTPUT FUZZY SETS	54

4.2. WALL HUGGING LAYER	56
4.2.1. RULE BASE	57
4.3. OBSTACLE AVOIDANCE LAYER	59
4.4. SUMMARY	60
5. <u>FPGA MODULES</u>	<u>61</u>
5.1. SENSOR MODULE	62
5.2. MOTOR DRIVER MODULE	69
5.3. SUPERVISOR MODULE	71
5.4. SUMMARY	76
6. <u>RESULTS</u>	<u>77</u>
7. <u>APPENDIX</u>	<u>80</u>
8. <u>BIBLIOGRAPHY</u>	<u>149</u>

LIST OF FIGURES

<i>Figure 2.1 Comparison between Conventional Set and Fuzzy Set.....</i>	<i>11</i>
<i>Figure 2.2 Complement of a Fuzzy Set.</i>	<i>13</i>
<i>Figure 2.3 Union of two Fuzzy Sets.....</i>	<i>14</i>
<i>Figure 2.4 Intersection of Fuzzy Sets A and B.</i>	<i>15</i>
<i>Figure 2.5 Fuzzy Partitioning of space U.....</i>	<i>16</i>
<i>Figure 2.6 Example of Linguistic Variables.</i>	<i>17</i>
<i>Figure 2.7 Block Diagram of a Fuzzy Logic Controller.</i>	<i>19</i>
<i>Figure 2.8 Example of Fuzzification.</i>	<i>20</i>
<i>Figure 2.9 Min-Max Method of inference to produce fuzzy sets.....</i>	<i>22</i>
<i>Figure 3.1 Ultrasonic Sensor SRF04.....</i>	<i>26</i>
<i>Figure 3.2 Timing Diagram of Ultrasonic Sensor.</i>	<i>28</i>
<i>Figure 3.3 Pin-out for SRF04 sensor.....</i>	<i>29</i>
<i>Figure 3.4 3.5" Sullivan Wheels fitted with encoder disc.....</i>	<i>31</i>
<i>Figure 3.5 Hamamatsu P5587 infrared photo-reflector sensor.....</i>	<i>32</i>
<i>Figure 3.6 R1 robot platforms.....</i>	<i>33</i>
<i>Figure 3.7 Block diagram representation of the overall robot system.</i>	<i>35</i>
<i>Figure 3.8 Architecture of FPGA.</i>	<i>36</i>
<i>Figure 3.9 Architecture of CLB.....</i>	<i>37</i>
<i>Figure 3.10 Input/Output Block (IOB)</i>	<i>38</i>
<i>Figure 3.11 Virtex-II Pro LC Development Kit.....</i>	<i>41</i>
<i>Figure 3.12 Block diagram of a Virtex-II Pro LC development kit.....</i>	<i>42</i>

<i>Figure 3.13 MOC3010 schematic.....</i>	<i>44</i>
<i>Figure 3.14 Connection diagram for 5V to 3.3V converter.....</i>	<i>45</i>
<i>Figure 3.15 2N3904 as an inverter.....</i>	<i>46</i>
<i>Figure 3.16 3.3V to 5V converter circuit.....</i>	<i>46</i>
<i>Figure 4.1 Schematic of Layered Motion Control System for robot navigation.....</i>	<i>49</i>
<i>Figure 4.2 Fuzzy sets for error input.....</i>	<i>51</i>
<i>Figure 4.3 Fuzzy sets for the Rate of Change of Error input.....</i>	<i>52</i>
<i>Figure 4.4 Output fuzzy sets for centering mode.....</i>	<i>55</i>
<i>Figure 5.1 A 'TRANSCIEVER' Block.....</i>	<i>62</i>
<i>Figure 5.2 Timing for a 'TRANSCIEVER' block.....</i>	<i>63</i>
<i>Figure 5.3 'TRIGGER' block</i>	<i>64</i>
<i>Figure 5.4 'LATCH' block.....</i>	<i>65</i>
<i>Figure 5.5 Interconnection of all the blocks.....</i>	<i>66</i>
<i>Figure 5.6 'RESET' block.....</i>	<i>67</i>
<i>Figure 5.7 Block diagram of the Sensor Module.....</i>	<i>68</i>
<i>Figure 5.8 Block diagram for motor driver module.....</i>	<i>69</i>
<i>Figure 5.9 Block diagram of the supervisor module.....</i>	<i>71</i>
<i>Figure 5.10 Interface block for the LCD screen.....</i>	<i>72</i>
<i>Figure 5.11 Block diagram showing the interconnection of various modules.....</i>	<i>74</i>
<i>Figure 5.12 Block diagram of overall navigational controller for the robot.....</i>	<i>75</i>
<i>Figure 6.1 Schematic of robot environment.....</i>	<i>78</i>

LIST OF TABLES

Table 4.1 Rule base for centering mode. 54

Table 4.2 Rule Base for Wall Hugging mode.58

ACKNOWLEDGEMENTS

I would like to acknowledge all the individuals who have helped me to complete this thesis and provided the guidance whenever I needed. Without the support of my thesis director Dr. Kuldeep S. Rattan I would not have been able to reach my goals. Many thanks to him, as he showed me the right direction and guided me during difficult situations. This thesis would not have been possible without his support. I would like to express my gratitude to him for his guidance not only in my thesis, but all throughout the course of my Master's program.

I would like to thank Dr. John M. Emmert and Dr. Pradeep Misra for evaluating my thesis and their interest and support during my defense. I would also like to acknowledge the assistance provided by the Electrical Engineering department with a special note of thanks to Sara Hohne and Vickie Slone for the constant help that they provided.

Finally, I thank my mother Surinder Kaur, father Nirmal Jit Singh Khatra and my family members for all their love, support, constant encouragement, and showing the right path in life.

Chapter 1

Introduction

The need of increased productivity and lower labor costs for manufacturing units has resulted in increased research interests in the field of automation and robotics. Industries are pumping in huge sums of money in their R&D department to develop more sophisticated machines to cut the manufacturing costs. As a result many new technologies like, radio frequency identification (RFID), Global Positioning Systems (GPS), embedded firmware have evolved which have helped to boost the manufacturing processes. New robotic innovations have overtaken tasks from human beings that are considered to be difficult, dangerous or dull, since the robots can perform the same task over and again without making any errors and getting bored. The development of new technologies like robots is inevitable, and in the near future a large number of human tasks will be overtaken by the robots, thus making human life a lot easier, and safer.

A robot can be considered as a computer with a mechanical body, which not only stores data or performs complex computations with its brain but also uses its mechanical body to perform physical tasks. Many new robots have been developed to serve humans. They come in different shapes and perform different tasks but can be divided into two main categories of industrial robots and mobile robots. The industrial robots facilitate the manufacturing processes like welding, painting, manufacturing etc. These robots have

their base platform fixed at a point and the job that is to be processed is brought to the robot by using conveyors i.e. these robots do not have any mobility. Some of the companies that manufacture industrial robots are Yasakawa, Mitsubishi, Fanuc, ABB, Nachi etc.

On the other hand, mobile robots are equipped with an additional feature of mobility. They are not equipped to perform complex mechanical tasks like the industrial robots, because their platform can be loaded to a reasonable extent to assist mobility. A heavier platform is harder to control and would lack in smooth navigation. In their case they can travel up to the job and process it, without having the need to move the job to the robot. These mobile robots can be further classified as Automatically Guided Vehicles (AGVs) and Autonomous Mobile Robots (AMRs). Automatically Guided Vehicles are guided by the use of a peripheral guide like a wire to stay within their work area. Inside this work area they can have a random movement or can be guided further in a specific direction by the use of a remote control. A common example of AGV is RoboMower [4], a robotic lawn mower which is guided by a buried wire to ensure it remains in the boundaries. The problem with AGVs is that they have a limited flexibility. If for any reason they enter a state which is not programmed into them, human intervention is needed to reset the robot for its normal functioning.

Autonomous Mobile Robots are designed with a main objective of them being autonomous i.e. they should not need any human intervention at any period of time and they should be able to make their own decisions if they enter into an unknown state. The

AMR is capable of working in any environment with the help of a number of sensors and a strong processing power. It takes over the tasks done by its human counterparts, and is capable of traversing in an unknown environment by sensing the physical patterns, making memory maps, and learning from experience. This enables the humans to hand over the tasks which are dangerous or require more speed and strength. The robots can work over and over again on the same task without getting tired, work in a toxic environment which is inaccessible by human beings.

With the innovation of technologies like RFID, Autonomous Mobile Robots are used to keep a track of the inventory in an industrial plant. Some other tasks that are performed by the AMRs include maintenance, inspection, security, repair work etc at places which are inaccessible to humans. A commercially available AMR is PeopleBot [7], which is used for guided tours of building complexes.

1.1 Background

Autonomous Mobile Robots are designed with one main goal that they should not need human intervention. To achieve this, these robots are equipped with a multitude of sensors and ample processing power. The algorithm that controls the robot navigation should be sophisticated enough to guide the robot to its goal frame. Many navigation algorithms like hardwired control and traditional control model have been implemented for the robot control. These algorithms are based upon the exact mathematical model of the robot. Their accuracy to control the robot depends upon the preciseness of the

mathematical model. However, the changing model causes some change in the parameters of the model like friction, wheel slippage etc. Moreover the environmental conditions keep changing continuously, making it all the more difficult to control the robot using the traditional control techniques. To overcome all these shortcomings, fuzzy navigational control scheme is used. This algorithm does not depend on the mathematical model of the robot, and hence is less prone to errors.

1.2 Problem Description

The main objective of this thesis is to develop an AMR which is capable of traversing through changing environments like hallways and passageways and achieve its goal frame without crashing into any of the obstacles. The algorithm used to control the robot for effective navigation is layered fuzzy logic controller. This navigation algorithm is divided into different layers to keep the algorithm simple and have a better control of the robot. Each layer in the navigation algorithm simulates the behavior of the robot in different environments.

The three layers used by the navigation algorithm are: Orientation Layer, PD Layer, and Obstacle Avoidance Layer. The Orientation Layer helps the robot align itself in the direction of the goal frame and takes over the control of the robot when the robot does not sense any obstacle in the path. It helps the robot to point towards the goal frame and the robot moves in the direction of its destination. This layer keeps a track of the distance traveled by the robot, and its distance from the goal frame. As the robot moves in the

direction of the goal, it keeps on updating these records, and computes the new direction and distance for the goal frame. This layer is active only if the other layers are dormant.

The second layer, PD Layer, helps the robot navigation through passageways. Depending upon the different kinds of passages, this layer is further divided into three modes: centering mode, wall hugging mode, and the blind mode. If the robot senses walls on both sides of the hallway, the robot follows its centering mode. In this mode, the robot tries to come into the center of the passage and follow the centering mode until either or both of the walls cease to exist. The Wall Hugging mode is used if the robot senses wall on only one side of the passageway. In this mode, the robot tries to move at a fixed distance from the wall. If the walls cease to exist on both sides of the passageway, then the robot follows its blind mode. Here the robot simply checks for any obstacles. If it does not sense any, it continues its forward motion in a straight line.

The third layer, Obstacle Avoidance layer helps the robot move away from any obstacles in the path as it navigates through dynamically changing environment and hence avoiding a crash. This layer is invoked if the sensors sense an object whose proximity from the robot base is less than a preset value. In such a situation the obstacle avoidance layer assists the robot move in a direction to stay at a safe distance from the obstacle.

This multi-layered navigation algorithm is implemented on a robotic platform which is built on modular approach consisting of three modules: Sensor module, Motor driver module, and the supervisor module. The sensor module consists of two ultrasonic sensors

on the left and right sides and one infrared sensor on the front end of the robot base. This module senses the obstacles in the vicinity of the robot and computes its distance from the robot platform. The motor driver module assists the robot in moving in a specific direction. It also keeps a record of the distance traveled by the robot. The supervisor module acts as the brain of the robot and makes all the decisions regarding the navigation strategy for the robot system. This module holds the multilayered navigational algorithm and is implemented on an FPGA [13].

The sensor and the motor driver modules which include the ultrasonic sensors and the servo motors require electrical signals for their operation. The ultrasonic sensor needs a trigger pulse to start their sensing operation. Similarly, the servo amplifiers in the motors need PWM signals for speed and direction control. These signals are provided by the FPGA. This is done by dividing the FPGA into modules that are dedicated for the control of the sensors and the motors.

The advantages of FPGA implementation over the microcontroller implementation of the layered fuzzy controller are a compact design and the elimination of inter-module communication over different platforms which results in an improved navigation in dynamically changing environments. The FPGA has faster clock speeds which improves the reaction time of the robot. It also provides possibilities of dynamic reconfiguration.

1.3 Outline of Thesis

Chapter 2 explains the basics of fuzzy logic theory and provides a functioning of the fuzzy logic controller. Chapter 3 describes the hardware involved in the sensor, motor driver and the supervisor module. The layered fuzzy motion controller is discussed in chapter 4. Chapter 5 provides an insight into the FPGA implementation of the navigational algorithm. The results of implementation of the layered motion controller on the robot system are discussed in chapter 6. Chapter 7 provides the conclusions and recommendations for future work followed by appendices and bibliography.

Chapter 2

Introduction to Fuzzy Logic

Recent developments in different fields of technology show an increased inclination towards digitization. This has resulted in improved performance and better service, but there is still a huge part played by the analog world in our lives. The main difference between the two is that in digital logic, the transition from one state to another is sharp and the boundary limits are strictly defined, whereas in case of analog logic the transition from one state to another is smooth and there is a gradual transition between the boundary limits. Human thinking is one good example of processing information using analog logic. We use linguistics to define a physical quantity like temperature as cold, warm, hot or very hot. We do not say that an object is cold if it is at 19°C and it is hot if it is at 22°C . We would rate these two temperatures in the same category or in categories very near to each other. Fuzzy logic simulates this way of human thinking to process the information. It was conceived by Lofti Zadeh, a professor at The University of California at Berkley and presents a way of processing information by allowing partial set membership rather than crisp set membership.

Fuzzy Logic is being used for the control of various systems. The main advantage of fuzzy logic is that it does not need the mathematical model of the plant to be controlled. Because of this, fuzzy logic can be used to control systems where the mathematical model of the plant to be controlled is too complex to determine, or the plant parameters

vary with time, changing the mathematical model for the plant. Fuzzy logic controllers are more robust than the PID controllers since they cover a wider range of operating conditions and provide a better control of the systems in the presence of noise and disturbances. Moreover, the fuzzy logic controllers are easier to understand as compared to the traditional PID, hence making them more customizable and easier for future modifications. The development of a fuzzy logic controller is cheaper than a model based PID controller for the same task.

Fuzzy Logic can be explained by taking an example of a car traveling on a highway at 63 mph. Now if the speed limit is 60 mph, a traffic policeman would not give a ticket to this car for traveling over by 3 mph. But the driver would surely get a ticket if he is traveling at 70 or 75 mph. In this example, the traffic policeman checks the speed of the traveling car, and then analyzes it by using linguistics like slightly over, over, too much over. The traffic policeman then decides to give a ticket depending upon these analyzed linguistics. Fuzzy logic operates on these same fundamental lines. It is divided into three steps. First, the input variable is taken in and converted in linguistic variable such as fast, slow etc. This process is called fuzzification. In the second step the fuzzy output is determined depending upon the linguistic variable and the rule base. This is called inference evaluation. The last step is to convert the fuzzy output into a crisp value for use by the real world.

2.1 Basic Elements of Fuzzy Logic

This section explains the basic elements that make up the theoretical part of the fuzzy logic. These elements form the basic structure required for the fuzzification, inference evaluation, and the defuzzification process. Fuzzy logic theory encompasses fuzzy sets, fuzzy conditional statements, linguistic variables, fuzzy partitioning, and the knowledge base. These are discussed in detail in the following section

2.1.1 Fuzzy Sets

Traditional control theory allows a given condition to be either true or false. In other words a condition can have a value of either 0 or 1 for a given set in the universe of discourse, and the transition from one set to another is sharp. Fuzzy logic, on the other hand, allows this transition to be smooth and the membership to a set lies between a value of 0 and 1 and is called a fuzzy set. Therefore, a fuzzy set is a generalization of a classical set by allowing the membership function to take any value between 0 and 1. The difference between a classical set and a fuzzy set is best illustrated by figure 2.1 [5]. The two sets depict age near 40. As can be seen from the figure that the conventional set has abrupt change from membership to non membership whereas this change is more gradual in the case of a fuzzy set.

Figure 2.1 Comparison between Conventional Set and Fuzzy Set.

Hence, a fuzzy set in the universe of discourse U is characterized by a membership function $\mu(x)$ that takes the values in the interval 0 and 1. A membership function $\mu(x)$ can have various shapes like triangular, trapezoidal, quadratic, Gaussian etc depending upon the application to be controlled and can be determined by using the data collected from the sensors or by using the knowledge of a human expert. These details give the basic shape for the membership function but a minor fine tuning is required for appropriate representation of different systems. Triangular and trapezoidal are two examples of piecewise linear functions and are more popular for use as membership functions because of simplicity of calculations. However, extra smoothness in the membership functions can be introduced by the use of quadratic or Gaussian functions but the complexities involved in calculations increases and hence are sparingly used.

2.1.2 Fuzzy Set Terminology

Consider an element x in the universe of discourse U such that $x \in U$, then there exists a membership function $\mu_A(x)$, which depicts the membership of element x in the fuzzy set A which takes on the values between 0 and 1. The following concepts are used in the fuzzy set theory:

- Support: The support of a fuzzy set A is a collection of all the elements in the universe of discourse U which have a non zero membership for the fuzzy set A i.e. $\mu_A(x) > 0$
- Fuzzy Singleton: A fuzzy singleton is a fuzzy set whose support value is a single point in the universe of discourse U .
- Crossover Point: Cross-over point of a fuzzy set A is an element x such that the membership function $\mu_A(x)$ has a value of 0.5.
- Height of a Fuzzy Set: The maximum value of membership attained in a fuzzy set A is known as the height of fuzzy set A .
- Center of a Fuzzy Set: It is the mean value of all the points at which the membership function $\mu_A(x)$ achieves its maximum value.

2.1.3 Fuzzy Set Operators

This section describes the basic operators that can be used along with the fuzzy sets. The fuzzy sets can be operated upon in a way similar to the conventional sets. Consider two fuzzy sets A and B and their membership functions to be $\mu_A(x)$ and $\mu_B(x)$. The basic operations that can be applied are summarized as follows

- Complement: The complement of a fuzzy set A can be defined by point wise membership function such that

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

where $\mu_A(x)$ is the membership function for fuzzy set A and $\mu_{\bar{A}}(x)$ is the membership function of the complement. This operation is similar to the logical NOT operation used for manipulation of the digital signals. This operation can be graphically represented as [9]

Figure 2.2. Complement of a Fuzzy Set.

- Union: The union of a fuzzy set A and a fuzzy set B is defined by a membership function such that

$$\mu_{A \cup B}(x) = \mu_A(x) \cup \mu_B(x) = \max(\mu_A(x), \mu_B(x))$$

where $\mu_{A \cup B}(x)$ is the membership function of the resultant fuzzy set after the union operation, $\mu_A(x)$ is the membership function of fuzzy set A and $\mu_B(x)$ is the membership function of the fuzzy set B. The union operation can be graphically represented as [9]:

Figure 2.3. Union of two Fuzzy Sets.

The dark lines in Figure 2.3 represent the union of the two fuzzy sets A and B. As can be seen from this figure that the union operation is similar to the logical OR operation or the max operator of the fuzzy sets.

- Intersection: The intersection of a fuzzy set A and a fuzzy set B is defined by a membership function such that

$$\mu_{A \cap B}(x) = \mu_A(x) \cap \mu_B(x) = \min(\mu_A(x), \mu_B(x))$$

$$\text{or, } \mu_{A \cap B}(x) = \text{prod}(\mu_A(x), \mu_B(x))$$

where $\mu_{A \cap B}(x)$ represents the membership function of the resultant fuzzy set after the intersection operation, $\mu_A(x)$ is the membership function of fuzzy set A, and $\mu_B(x)$ is the membership function of fuzzy set B. The intersection operation can be graphically represented as

Figure 2.4 Intersection of Fuzzy Sets A and B.

As can be seen from figure 2.4 that the intersection operation can be carried out using either the min operator or the prod and they produce different results. The dark line in this figure represents the resulting membership function after the intersection operation. Intersection using min operator involves less computational complexity as it requires the comparison of two membership functions and selecting the smaller value. On the other hand prod operator requires the multiplication of the two membership values to compute the resulting

membership functions. The present thesis work uses the min-max approach for computational purposes.

2.1.4 Fuzzy Partitioning

Fuzzy sets and membership function allow a smooth transition from membership to non membership, whereas in case of a classical partition this change is abrupt. A fuzzy partition generalizes this concept and allows a gradual transition at the time of exit from a fuzzy set. In other words, as the membership value in fuzzy set A_1 reduces while traversing the subspace, the membership value for another fuzzy set A_2 increases. This change is smooth as the point X gradually moves out of A_1 . Hence the fuzzy partitions of space U for a variable X are the collection of fuzzy sets A_i , which satisfies the following condition

$$\sum_{i=1}^n \mu_{A_i}(x) = 1$$

for all $x \in U$. The fuzzy partitioning for the universe of discourse U can be graphically represented as

Figure 2.5. Fuzzy Partitioning of space U .

2.1.5 Linguistic Variables

Linguistic variables are used in conjunction with fuzzy sets. When a variable takes numbers as its values, a mathematical framework can be established to formulate it. This framework can be used to calculate the value of the variable under different conditions. However, when a variable takes words in natural language as its values, it is called as a linguistic variable and a mathematical framework cannot be established for computational purposes. For example, the speed of a car can be described by numerical values where a mathematical framework can be constructed to describe the variable at different instances. However the speed of a car can also be described as slow, medium, or fast. In such a case, speed acts as a linguistic variable and is characterized by fuzzy sets. This can be graphically shown as [9]

Figure 2.6. Example of Linguistic Variables.

2.1.6 Fuzzy Control Rules

Fuzzy control rules act as connectors between the inputs and outputs. These rules are used to compute the fuzzy output for a given fuzzy input. These rules act as the backbone for the fuzzy control system. These rules are formulated using the knowledge from the human expert or based on the data collected from the sensors. These rules are a collection of “if then” statements that govern the output from the fuzzy controller. Since these rules are made up using common linguistics, they are easier to understand and can be easily modified. These statements have an antecedent as the inputs and the consequent result is the resultant control action.

2.2 Fuzzy Logic Controller

Traditional control model has been used successfully to control systems where an accurate mathematical model of the plant is available and the plant parameters remain the same under approximately all conditions. However, in a case where the mathematical model of the plant is available with a smaller degree of precision or the plant parameters change under the ambient conditions, the traditional control model fails to provide an adept control of the system. Under such a scenario, the mathematical model of the plant is replaced by a knowledge base made up of fuzzy sets, linguistic variables, and the fuzzy rules. The resulting control model is known as a fuzzy logic controller. The knowledge base is built by using the knowledge of the operator, or sensor data. Such a control system has proven to provide an improved control of the system. The block diagram emulating a fuzzy logic controller is shown in figure 2.7 [9].

Figure 2.7. Block Diagram of a Fuzzy Logic Controller.

As can be seen from figure 2.7 that the operation of a fuzzy logic controller can be divided in three steps: Fuzzification, Inference Evaluation, and Defuzzification. These three steps are explained in the following section.

2.2.1 Fuzzification

The input to a fuzzy logic controller is a crisp input, which has a specific value representing a real world variable, whereas a fuzzy logic controller deals with linguistic variables and fuzzy sets. Hence this crisp value needs to be converted into a membership value for the input fuzzy sets. This process of converting a crisp value into a membership value for the input fuzzy sets to be used by the fuzzy logic controller is called fuzzification. The input to the fuzzification interface is a crisp value and the output is the degree of membership for the input fuzzy sets. This can be explained with the help of figure 2.8.

Figure 2.8. Example of Fuzzification.

In the given example, a speed of 70 mph is fuzzified into sets of medium and fast. As can be seen from this example the speed of 70 mph has a membership of 0.75 in the fuzzy set 'medium' and a membership of 0.25 in the fuzzy set 'fast'. These membership values are

used by the fuzzy logic controller to determine the output fuzzy sets by using inference evaluation.

2.2.2 Inference Evaluation

The process of determining the fuzzy output from the input fuzzy sets is called inference evaluation. The process of inference evaluation is accomplished by using the knowledge base. The knowledge base of a fuzzy logic controller consists of linguistic variable definitions and if-then rules. The rules are formulated using the knowledge of the human experts or sensor data. These two elements determine how the controller will behave. The knowledge base helps to determine the output fuzzy sets. The shape of the output fuzzy set can be determined by using min-max method inference method and is illustrated in figure 2.9

Figure 2.9. Min-Max Method of inference to produce fuzzy sets.

The following example extends the previous fuzzification example and the fuzzy controller computes the amount of breaking power required to stop a car traveling at 70 mph. The knowledge base for this control system has the following if then rules:

- If the speed of the car is slow then use light brakes,
- if the speed of the car is medium then use moderate braking power and,
- if the speed of the car is fast then use strong braking power.

Using these rules, the output of the controller is determined using the min-max method. The output fuzzy set for moderate speed is shown in figure 2.9(a). The output for fast speed is shown in figure 2.9 (b) and the resultant fuzzy set which is the combination of the maximum of the two fuzzy sets is shown in 2.9 (c).

2.2.3 Defuzzification

Since a fuzzy logic controller controls a real world system, the output from the controller should be a crisp value which can be fed as an input into the system. This process of converting the fuzzy output from the inference engine into a crisp value is called defuzzification. Various methods are proposed for defuzzification process, but they produce a little practical difference. One of the most popular methods used for defuzzification is the weighted average method. The crisp value is calculated by using the formula

$$U = \frac{\sum_{i=1}^k \mu_{A_i}(x) \cdot \omega_i}{\sum_{i=1}^k \mu_{A_i}(x)}$$

where U is the crisp value to be used by the real world system, $\mu_{A_i}(x)$ is the membership in the A_i^{th} fuzzy set, and ω_i is the center value of the corresponding output fuzzy set. In

this example $\mu_{A1} = 0.75$, $\mu_{A2} = 0.25$, $\omega_1 = 60$, and $\omega_2 = 85$. Using these values the crisp value for the output is calculated as

$$U = \frac{0.75 * 60 + 0.25 * 85}{0.75 + 0.25} = 66.25$$

This thesis uses the matrix approach for the fuzzy logic controller which utilizes the min-max approach for inference evaluation, and weighted average method for defuzzification process.

2.3 Summary

This chapter describes the basics of a fuzzy logic providing an insight into the fuzzy sets, its terminology and operators, fuzzy partitioning, linguistic variables and fuzzy control rules. It also explains the basic structure of a fuzzy logic controller and its implementation. The next chapter describes the hardware involved in the various modules of autonomous mobile robot system.

Chapter 3

Hardware Description

This thesis aims at the navigational control of an autonomous mobile robot [3]. The robotic platform used for the thesis is built on a modular approach i.e. the platform is divided into different modules. The main advantages of the modular approach are

- *Decentralization of Control:* The modular approach helps to distribute the control of the robot amongst different modules. Each of these modules is enabled with its own data processing power, and share the load of the supervisor module. This enables the supervisor module to focus on just the navigational control of the robot. The processing required for controlling the sensors, and the motors is taken care of by their respective modules.
- *Future Enhancements:* The modular approach facilitates the process of future enhancements of the robotic platform. The older modules can be replaced by more powerful newer modules, without the need to build a new robot. The modular approach enables the basic structure of the robot to be generic and allowing different versions of robots to be built on this structure, satisfying different requirements.
- *Facilitates Fault Detection:* The modular approach facilitates the fault detection process. Each module can be tested individually for correct operation, and the faulty module can be replaced by a newer one.

The autonomous mobile robot is divided into three modules: Sensor module, Motor Driver module, and the Supervisor module. These modules are explained in the following section.

3.1 Sensor Module

The sensor module for the robot consists of the sensors and the software code inside the FPGA (see chapter 5) controlling these sensors. Various sensors are available in the market like infrared sensors, ultrasonic sensors, laser beam sensors, etc. The sensors selected depend on the application requirements. The robotic platform for this thesis uses four ultrasonic SRF04 sensors for sensing the distance of the obstacle. It has a compact design and gives a range of 3 cm to 3 m. Figure 3.1 shows the picture of the SRF04 ultrasonic sensor [6].

Figure 3.1 Ultrasonic Sensor SRF04.

The specifications of ultrasonic sensor are given as follows

- *Voltage*: 5 V
- *Current*: 30mA Typ. 50mA Max.
- *Frequency*: 40 KHz.
- *Maximum Range*: 3 m.
- *Minimum Range*: 3 cm.
- *Sensitivity*: Detect a 3cm diameter stick at greater than 2 m.
- *Input Trigger*: 10uS Min. TTL level pulse.
- *Echo Pulse*: Positive TTL level signal, width proportional to range.
- *Weight*: 0.4 oz.
- *Size*: 1.75" w x 0.625" h x 0.5" d.

The VHDL code in the FPGA controls the sensors on the robot platform by providing them with a trigger pulse to start the sensing operation. The SRF04 sensors require a trigger input for a minimum period of 10 μ sec. After receiving the firing pulse, the sensor sends out a sonic burst for 8 cycles to sense an obstacle in the robot proximity. The sensor pulls the echo pulse output to high after the sonic burst and remains in this state until it detects the reflected sonic bursts. The distance of the obstacle from the robot base is directly proportional to the time span for which the echo pulse remains high and can be calculated by using the following formula:

$$Dist = \frac{Speed_of_sound * Time}{2}$$

If the sensor does not detect the reflected sonic burst, the echo pulse output goes low after a period of 18msec. The timing diagram for SRF04 sensor is shown in figure 3.2 [6]

Figure 3.2 Timing Diagram of Ultrasonic Sensor.

As can be seen from the timing diagram, the SRF04 sensor has the firing pulse as an input and echo pulse as an output. At the end of one sensing cycle when the echo pulse goes into a low state the sensor requires a 10 m sec delay before the next sensing cycle starts. If a trigger pulse is applied before the delay period for the sensor, it fails to respond. The SRF04 sensor also requires supply connection of 5V and the pin out for the sensor is shown in figure 3.3 [6]

SRF04 Connections

Figure 3.3 Pin-out for SRF04 sensor.

The sensor module acts as an interface between the supervisor module and the sensors on the robot. It takes the instructions from the supervisor module regarding the sequence for sensor operation, provides trigger pulses to the sensors according to the given sequence, and keeps a record of the distance calculations for each sensor. This database is used by the supervisor module for future decision making process. The VHDL code for the sensor module operation is discussed in chapter 5.

3.2 Motor Driver Module

The motor driver module for the robot consists of the motors, wheels, encoder, and the control circuitry for direction and speed control. Two high speed FUTABA servo motors [14] are used for moving the robot platform. The speed and direction of these motors is controlled by using pulse width modulation. The specifications for the servo motors are as follows:

- *Control System:* +Pulse Width Control 1520usec Neutral
- *Required Pulse:* 3-5 Volt Peak to Peak square wave.
- *Operating Voltage:* 4.8 Volts
- *Operating Temperature range:* -20 to +60 Degree C
- *Operating Speed (4.8V):* 0.21 sec/60 degrees at no load
- *Stall Torque (4.8V):* 128 oz/in.
- *Operating Angle:* 45 Deg. one side pulses traveling 400usec.
- *360 Modifiable:* yes
- *Direction:* Counter Clockwise/Pulse traveling 1520-1900usec
- *Motor type:* 3-Pole motor
- *Potentiometer Drive:* Indirect Drive
- *Bearing Type:* Dual Ball Bearing
- *Connector Wire Length:* 12"
- *Dimensions:* 2.4" x 1.2" x 2.0"
- *Weight:* 3.4 oz

The shafts of the two servo motors are connected to two rubber wheels 3.5" in diameter. The inner surface of these rubber wheels is fitted with a wheel encoder disc which is slotted with 44 black and white stripes for the encoder to determine the wheel displacement and the velocity. Figure 3.4 [14] shows the picture of the Sullivan wheels used on the robot rear end.

Figure 3.4 3.5" Sullivan Wheels fitted with encoder disc.

The front end of the robot consists of 2" diameter Sullivan wheels. The motor driver module uses Hamamatsu P5587 wheel encoder module pair. This pair of wheel encoder modules provides a no soldering solution for sensing robot wheel position and velocity. These modules are very simple and compact, and have a 3-pin (0.100" spacing) header. They use the low voltage Hamamatsu P5587 infrared photo-reflector sensor to sense black to white transitions on a wheel. These modules require +5V and GND to power them, and provide a 0 to 5V square wave output (+5V output when they "see" white, and a 0V output when they "see" black). The control mechanism can simply read the 0-5-0V pulse train produced from the white to black transitions on the wheel to determine how far and how fast the robot has traveled. Figure 3.5 shows the picture of Hamamatsu P5587 infrared photo-reflector sensor. [14]

Figure 3.5 Hamamatsu P5587 infrared photo-reflector sensor.

The control circuitry for the motor driver module consists of logic to generate a pulse width modulation output for the servo motors, to control their speed and direction of rotation. This logic is also responsible for counting the pulses from the encoders to determine the distance covered by the robot, and the speed with which the robot is traveling. It maintains a database of this information for use by the supervisor module. The supervisor module uses this information for making decisions regarding navigational control of the robot. The logic part of the motor driver module is implemented as a VHDL software code on an FPGA discussed in chapter 5.

The hardware part of the motor driver module and the sensor module consisting of the sensors, servo motors, wheels, encoders, and the robot base are integrated together to form a single unit. The motors, wheels, encoders, and the robot base are commercially available as parts of R1 robot. Figure 3.6 shows the picture of R1 robot [14].

Figure 3.6 R1 robot platforms.

The R1 (1-level) measures 9 3/8" high x 11 5/8" wide x 11 5/8" deep. It can carry up to 10 lbs of load and is a modular platform. The ultrasonic SRF04 sensors are mounted on the four corners of the robot platform. The supervisor module and the interface circuitry for interconnecting these modules are placed on the platform base.

3.3 Supervisor Module

The supervisor module is dedicated for the navigational control of the robot platform. It acts as the brain for the robot and provides an overall control for the robot modules. The various tasks performed by the supervisor module are

- Send instructions to the sensor module regarding the firing sequence of the sensors.
- Poll the sensor regarding the distance values of the obstacles in the path of the robot.
- Depending on these distances, compute the turn angle for the robot.
- Convert the turn angle of the robot into speed and direction for the left and right motors.
- Communicate with the motor driver module to get the distance traveled by the robot platform.

The supervisor module, like the motor driver and sensor modules, comprises of hardware and software. The software part deals with the computation of the turn angles for the robot and polling sequence generation for the ultrasonic sensors and is implemented as a VHDL code. The hardware, which is an FPGA, takes care of the signals required for PWM and firing signals for the ultrasonic sensors. A block diagram of the overall robot system is shown in figure 3.7.

Figure 3.7 Block diagram representation of the overall robot system.

3.3.1 Field Programmable Gate Array

A Field Programmable Gate Array (FPGA) is a programmable integrated circuit that is often used for prototyping digital circuits on site for a specific application. It utilizes gate array technology that can be reprogrammed after it is manufactured, rather than having its programming fixed during the manufacturing. The general architecture of FPGA is shown in figure 3.8.

Figure 3.8 Architecture of FPGA.

The FPGA architecture has three basic programmable components:

- Configurable Logic Block
- Programmable Input/Output Block
- Programmable Interconnect

3.3.1.1 Configurable Logic Block

The Configurable Logic Block (CLB) is the basic building block for an FPGA. The CLBs are arranged in an array for the implementation of the combinational and sequential logic designs. The general architecture of a CLB is shown in figure 3.9 [13].

Figure 3.9 Architecture of CLB.

A CLB architecture consists of two four input Look-up Table (LUT), along with logic gates, carry logic, multiplexers, and storage elements such as registers. The multiplexers facilitate the implementation of more than four input logic function. The carry logic helps to accelerate the arithmetic functions such as addition, subtraction, etc. The registers are configured as edge triggered D-flip flops.

3.3.1.2 Programmable Input/Output Block

The Programmable Input/Output Block (IOB) is a group of elements that configure the FPGA pins as either input or output for the digital design programmed in the FPGA. The IOB acts as an interface between the real world signals and the design configured in the FPGA CLBs. A simplified block diagram for the IOB is shown in figure 3.10.

Figure 3.10 Input/Output Block (IOB).

The IOB has storage devices such as edge sensitive D flip-flops or the level sensitive latches to detect the changes at the input pins for the FPGA.

3.3.1.3 Programmable Interconnect

A Programmable Interconnect is used to interconnect the various components of the FPGA like CLBs, IOBs, etc. This is done by using a number of metal segments with programmable switching points for interconnection. There are three main interconnect types:

- Single Length Lines
- Double Length Lines
- Long Lines

These lines can be either horizontal or vertical .The programmable interconnect resources can be either direct connections or general purpose with switch matrix. The direct connections are used to connect adjacent CLBs for fast signal transmission. General purpose interconnects are used to connect the horizontal and the vertical lines. This is generally done by using an SRAM cell.

3.3.2 Virtex-II Pro LC Development Board

The present thesis work utilizes a Virtex-II Pro FPGA for implementing the hardware of the supervisor module. This Virtex-II Pro FPGA comes along with other miscellaneous devices integrated onto a Virtex-II Pro LC Development board. The Memec Design Virtex-II Pro™ LC Development Kit provides a low cost and a complete development platform for designing and verifying applications based on the Xilinx Virtex-II Pro LC FPGA family. This kit enables designers to implement embedded processor based applications with extreme flexibility using IP cores and customized modules.

The Virtex-II Pro LC system board utilizes the Xilinx XC2VP4-5FG456C FPGA. The P4 FPGA contains a single PowerPC processor. The Virtex-II Pro LC board includes a 100MHz clock source, RS-232 port, USB2.0 to RS232-bridge, and additional user support circuitry to develop a complete system. The board also supports the Memec Design P160 expansion module standard, allowing application specific expansion modules to be easily added. Figure 3.11 shows the schematic of Virtex-II Pro LC Development Kit [13].

Figure 3.11 Virtex-II Pro LC Development Kit.

The Virtex-II Pro LC Development kit consists of the following features:

- Xilinx XC2VP4-FG456 FPGA
- 100MHz clock Oscillator
- LCD Panel
- F04S Platform Flash ISP PROM
- PC4 JTAG Programming/Configuration Port
- VisionProbe II Connection via CPU JTAG Port
- System ACE™ Interface Support
- RS232 Port
- USB2.0 to Serial Port
- Four User LEDs

- Eight User DIP Switch
- Four User Push-Button Switches
- P160 Module Interface
- Two 50-Pin User I/O Connectors, providing 64 user accessible I/O
- Piezo Speaker

The block diagram for development board is shown in figure 3.12.

Figure 3.12 Block diagram of a Virtex-II Pro LC development kit [13].

This FPGA module is placed on the robot platform along with the interface circuitry required for connecting the motor driver module and the sensor module to the FPGA.

3.4 Interface Circuits

Interface circuits are required for connecting the SRF04 sensors, motors, and the encoders to the FPGA board. The FPGA used for this thesis recognizes 0V as logic '0' and 3.3 V as logic '1', whereas the rest of the hardware, like sensors, motors, encoders etc on the robotic platform recognize 0V as logic '0' and 5 V as logic '1'. Hence there is a need to convert the 3.3 V output from the FPGA to 5V for the hardware input and 5V output from the sensors, and other equipment to 3.3V for FPGA input. Without the interface circuitry for this conversion, there is a risk of burning the FPGA. The interface circuitry used for interconnection purposes can be divided into two categories: 3.3V to 5V converters, and 5V to 3.3V converters. These two circuits are explained in the next section:

3.4.1 5V to 3.3V Converter

This circuit is used to connect the outputs from the sensors, and the encoders to the FPGA input pins. If this circuit is not used, the 5V output from the sensors is connected directly to the input pins of the FPGA which will result in burning the FPGA, since it is only $3.3V \pm 0.5$ tolerant. This circuit makes use of optical isolation between the sensor outputs and the FPGA inputs. Optical isolation has an advantage of isolating the voltage spikes from the sensor output from the FPGA input. The FPGA input pins are supplied a constant 3.3V input regardless of the voltage level on the output from the sensors. The

optical isolator used for the thesis is MOC3010. It is a 6 pin chip. The schematic of MOC3010 is shown in figure 3.13.

Figure 3.13 MOC3010 schematic.

The sensor output is connected to the anode (pin 1) through a 1 K resistor, and the cathode (pin 2) is grounded. A 3.3V constant supply is applied at pin 4 of the MOC3010 and the pin6 is connected to the input pins of the FPGA. Under normal conditions when the output from the sensor is low, pins 4 and 6 are isolated from each other and the input to the FPGA is 0V. However, when the output from the sensors goes high to 5V, the triac in the MOC3010 conducts and the voltage at pin 6 is applied to pin 4. Hence input pins of the FPGA are supplied by a constant 3.3V input. Figure 3.14 shows the connection diagram for this converter circuit.

Figure 3.14 Connection diagram for 5V to 3.3V converter.

3.4.2 3.3V to 5V Converter

This circuit is used to connect the output pins of the FPGA to the input of the sensors and the servo motor pulse width modulation inputs. If this circuit is not used the FPGA pins may be overloaded by the inputs of the sensors and may cause damage to the FPGA. This circuit uses a 2N3904 transistor in an inverter configuration. Figure 3.15 shows 2N3904 transistor in an inverter configuration. In this configuration; if an input of 0V is applied there is no current flow between the collector and emitter, and the output is pulled up to 5V by the pull-up resistor of the collector. However, when an input of around 5V is applied to the base of the transistor, there is a current flow between the collector and the emitter, and the output is grounded through the base and emitter. In the present case, the input to the inverter circuit ranges between 0-3.3V. The lower limit of the input voltage is consistent with the requirements. However, the upper limit of 3.3V is lower than the requirement of 5V. But connecting this input with the inverter circuit produces the same

results as those with 5V input and the output of the transistor goes low for an input of 3.3V.

Figure 3.15 2N3904 as an inverter.

Exploiting the fact that the inverter circuit recognizes a 3.3V input as logic '1' and produces an output of logic '0'; the output of the first stage of inverter is fed into another inverter to invert the output of the first stage. The overall circuitry for the voltage conversion is shown in figure 3.16.

Figure 3.16 3.3V to 5V converter circuit.

As can be seen from the circuit, for an input of 3.3V, the output of the first stage inverter is 0V. This is fed into the second inverter input and it produces an output 5V. For an input of 0V, the output from the overall circuit remains 0V. This circuit provides an efficient way of converting 3.3V into 5V as the base current which is the current drawn from the FPGA pins is very small and does not cause loading of the FPGA.

3.5 Summary

This chapter provided a detailed description of all the hardware involved in the implementation of the navigational control algorithm. The advantages of modular approach along with the different modules are described. The interface circuits were developed to connect the sensors and the motors to the FPGA. The next chapter describes the software aspect of the sensor and motor driver module. It explains the VHDL code that controls the sensor firing and computes the distances. The details for motor control using PWM are discussed later.

Chapter 4

Navigational Control Algorithm

This chapter describes the navigational algorithm used to control the robot propagation through passageways. This algorithm was proposed by Rodney Brooks [2] and later was proposed for implementation in fuzzy logic by Daniel Burke [1]. The original algorithm proposed by Brooks was generic and could be implemented using either the traditional control theory or the fuzzy logic control. But implementation of the algorithm using fuzzy logic controls has its own advantages and are discussed as follows:

- The traditional control theory depends upon the exact mathematical model of the robot being controlled for navigation. This means that each time this algorithm is used it needs to be supplied with a mathematical model of the robot. However, the implementation using fuzzy logic control does not have any requirements for the mathematical model of the robot. Hence, the implementation of the algorithm in fuzzy logic is model independent and is generic for a wide variety of robots.
- The traditional control model depends upon preciseness of the model. It does not take into account the errors caused by friction, wheel slippage etc. Hence, the control using the traditional model would lead to errors. However, these shortcomings are overcome by the fuzzy logic control of the robot.

- The motion controller using fuzzy logic is capable of responding to disturbances, changing environments, and imprecise sensory information.

This control algorithm is built in layers to control the robot at increasing levels of competence. The higher level layers can negate the control effect of the lower layers by suppressing their outputs. However, the lower levels continue to operate as higher level layers are added over them. This results in a flexible control system for robot navigation.

Figure 4.1 shows the schematic of the layered motion control scheme [12].

Figure 4.1 Schematic of Layered Motion Control System for robot navigation.

The layers take inputs from the various sensors on the robot platform as the distance of the object or walls, and produce the turn angle for the robot as an output as shown in fig 4.1. Each layer is dependent on a different set of sensors. For example, the Obstacle Avoidance layer is dependent on the front sensor, the centering layer is dependent on the left and the right sensors, and the wall hugging layer is dependent on either the left or the right sensor.

The matrix method is used for inference and defuzzification, where the inference engine uses the prod-sum method and the defuzzification is done using the weighted average method. The layers involved in the navigational algorithm are discussed in the following sections:

4.1 Centering Layer

This layer is used for navigational control of the robot through passageways with walls on both sides. In such a case this layer assists the robot in traveling through the middle of the passage. Since the robot has walls on both sides, this layer uses both the left sensor and the right sensor values. The inputs to the centering layer are the error, which is the difference between the left and the right sensor values, and the rate of change of error. These are calculated using the following formulas:

$$e(t) = L(t) - R(t)$$

and,

$$\dot{e}(t) = \frac{e(t) - e(t - t_s)}{t_s}$$

where $e(t)$ is the error input, $L(t)$ is the left sensor reading at instant ' t ', $R(t)$ is the right sensor reading at instant ' t ', $\dot{e}(t)$ is the rate of change of error input, t_s is the sampling time, $e(t - t_s)$ is the error value at the previous sampling instant. These inputs are fuzzified

using the input fuzzy sets for the error and the rate of change of error. The following section discusses the input fuzzy sets for these inputs.

4.1.1 Input Fuzzy Sets

The fuzzy sets for the error input are defined as: *negative big* (NB), *negative medium* (NM), *negative small* (NS), *zero* (ZO), *positive small* (PS), *positive medium* (PM), and *positive big* (PB) and are shown in figure 4.2 [12]

Fig 4.2 Fuzzy sets for error input.

Since the width of one passageway varies from another, the centers of the fuzzy sets for error input cannot be given a constant value. The centers for the fuzzy sets have to vary with the varying widths of different passages. This is done by making a generic analysis of the robot traveling through a passage. If we assume that L denotes the left sensor value, and R denotes the right sensor value, then the width the passageway is given by

(L+R) which is the maximum possible value for the error input. Since we have three fuzzy sets on the positive error axis and three fuzzy sets on the negative side, the error axis is divided into three equal partitions on each half. The centers for the fuzzy sets are assigned $\frac{1}{3} (L+R)$, $\frac{2}{3} (L+R)$, and $(L+R)$ for the positive side and $-\frac{1}{3} (L+R)$, $-\frac{2}{3} (L+R)$, and $-(L+R)$ for the negative side. Using this analysis, the centers of the fuzzy sets change dynamically with the varying widths of the passageways.

The fuzzy sets for the rate of change of error input are defined as: *negative big* (NB), *negative medium* (NM), *negative small* (NS), *zero* (ZO), *positive small* (PS), *positive medium* (PM), and *positive big* (PB). The centers for these fuzzy sets are fixed as [-3, -2, -1, 0, 1, 2, 3] and are shown in figure 4.3 [12]

Figure 4.3 Fuzzy sets for the Rate of Change of Error input.

Using these fuzzy inputs, a rule base is developed to determine the fuzzy output for the turn angle. The rule base is developed using the following analysis

4.1.2 Rule Base

The positive error input signifies that the left sensor value is greater than the right sensor value, making $L(t) - R(t)$ positive. If $R(t)$ is smaller than $L(t)$, the right wall is closer to the robot as compared to the left wall, or the robot is traveling closer to the right wall. Similarly, negative error signifies that the robot is traveling closer to the left wall. Similarly, a positive value of the rate of change of error means that the present value of error is greater than the error at the last sampling instant, or in other words the robot is traveling from left to the right. A negative value of rate of change of error indicates that the robot is moving from right to left. Based on these inputs, the rule base is constructed using the following logical reasoning:

- If the error value is positive and the rate of change of error is also positive, the robot is traveling closer to the right wall and the robot is moving to the right. In such a case, the output value of the turn angle should be such that the robot is turned towards the left wall. In other words the turn angle needs to be negative.
- If the error value is negative and the rate of change of error is also negative, the robot is traveling closer to the left wall and the robot is moving from right to the left. In this case, the robot should be instructed to turn to the right and hence the turn angle is positive.

- If the error value is zero and the rate of change of error is also zero the robot is traveling in the center of the passage and moving straight. Therefore, the turn angle for the robot is zero.

Based on the above discussion, the rule base for the centering mode is developed and is given in Table 4.1. [12]

Table 4.1 Rule base for centering mode.

		Error (e)						
		NB	NM	NS	ZO	PS	PM	PB
Change of error ($\dot{e}$)	NB	RB	RB	RB	RB	RM	RS	ZO
	NM	RB	RB	RB	RM	RS	ZO	LS
	NS	RB	RB	RM	RS	ZO	LS	LM
	ZO	RB	RM	RS	ZO	LS	LM	LB
	PS	RM	RS	ZO	LS	LM	LB	LB
	PM	RS	ZO	LS	LM	LB	LB	LB
	PB	ZO	LS	LM	LB	LB	LB	LB

4.1.2 Output Fuzzy Sets

The output fuzzy sets for the centering mode are defined as: *negative big* (NB), *negative medium* (NM), *negative small* (NS), *zero* (ZO), *positive small* (PS), *positive medium* (PM), and *positive big* (PB). The centers for these fuzzy sets are [-20, -15, -10, -5, 0, 5, 10, 15, 20] and are shown in figure 4.4. [12]

Figure 4.4 Output fuzzy sets for centering mode.

A positive value of the output fuzzy set means that the turn angle is towards the right, and a negative value indicates that the robot turn angle is towards the left. When the robot detects walls on both sides, the centering layer is activated and the robot moves in the center of the passage. However, when there is only one wall or the robot is unable to detect the second wall, the wall hugging layer is activated. The next section explains the wall hugging layer.

4.2 Wall Hugging Layer

The wall hugging layer is activated by the supervisor module if there is only one wall, or the robot fails to detect the second wall of the passage. This could happen if the passage is too wide to be covered by the range of the ultrasonic sensors. In such a case, the robot tries to move at a preset distance from the wall, and hence the name wall hugging layer. The wall hugging layer has only one input as the rate of change of error. The error input is set to zero because the reading from one of the sensors is always maxed out, and stays constant for robot motion. If the sensor fails to detect a wall on the right the robot moves parallel to the left wall and is called the left wall hugging mode. In the right wall hugging mode, the sensor fails to detect a wall on the left and the robot travels parallel to the right wall. The inputs for the left wall hugging mode are calculated by using the following formulas:

$$e(t) = 0$$
$$\dot{e}(t) = \frac{L(t) - L(t - t_s)}{t_s}$$

Similarly, for the right wall hugging mode the inputs are given by:

$$e(t) = 0$$
$$\dot{e}(t) = \frac{R(t) - R(t - t_s)}{t_s}$$

The input and output fuzzy sets for the wall hugging layer are the same as those of the centering mode with the only difference being that the error input is always zero and are shown in figures 4.3 and 4.4. The rule base for the wall hugging mode is discussed in the following section.

4.2.1 Rule Base

When the rate of change of error input is positive, the present value of the sensor is greater than the value at the last sampling instant. This signifies that the robot is moving away from the wall. On the other hand when the rate of change of error input is negative, the robot is moving closer to the wall. Based upon the physical significance of the input value the rule base is calculated using the following analysis:

- If the rate of change of error input is positive, it indicates that the robot is moving closer to the wall and the rectifying action should be such that the robot is moved closer to the wall. The direction of the turn angle output now depends upon the wall hugging mode. If the mode is left wall hugging, the turn angle is negative and if it is right wall hugging the turn angle is positive.
- If the rate of change of error input is negative, the robot is moving towards the wall. The action should direct the robot away from the wall. Depending upon the

wall hugging mode, the turn angle is positive for the left wall hugging and negative for the right wall hugging mode.

Based on the above discussion, the rule base for the wall hugging mode can be divided into two parts: rule base for left wall hugging mode and rule base for right wall hugging mode. The rule base developed for each wall hugging mode is shown in Table 4.2

Table 4.2 Rule Base for Wall Hugging mode.

		Left Wall Hugging mode	Right Wall Hugging mode
Rate of change of error	NB	RB	LB
	NM	RM	LM
	NS	RS	LS
	ZO	ZO	ZO
	PS	LS	RS
	PM	LM	RM
	PB	LB	RB

4.3 Obstacle Avoidance Layer

The Obstacle Avoidance Layer is used to prevent the robot from colliding with an object. This is done by placing a sensor on the front face of the robot. When an object comes in a preset proximity range of the robot, this layer is activated by the supervisor module. The preset proximity range for the robot is set to be 15 inches.

When the obstacle avoidance layer is activated, it takes over the control of the robot navigation and exercises a higher priority over the other layers. The outputs from the other two layers are forced to zero, and the turn angle output is controlled by this layer. When this layer is active, the forward motion of the robot is halted, and the robot makes a turn angle of 45° to steer past the obstacle safely.

For the present research work, the default direction of rotation is towards the right. However, if the robot detects another obstacle while turning right the direction of rotation is reversed to be left. The robot navigation is controlled by obstacle avoidance layer as long as the object is in the proximity range of the robot. After assisting the robot to steer past the object safely, the obstacle avoidance layer becomes dormant and the robot control is passed onto the centering layer or wall hugging layer depending upon the left sensor and the right sensor values.

4.4 Summary

This chapter describes a fuzzy layered motion controller for robot navigation through passages and hallways. This navigational algorithm is built using three elementary layers of control: Centering Layer, Wall Hugging Layer, and Obstacle Avoidance Layer. The details pertaining to the fuzzy sets and rule base are discussed for these layers. The next chapter discusses the FPGA implementation of the sensor, motor driver, and the supervisor module using the VHDL code and the XILINX ISE tool.

Chapter 5

FPGA Modules

The hardware for the robot platform is divided into modules to provide a better control and enhance the robot performance during navigation through hallways. The sensor and the motor driver modules which include the ultrasonic sensors and the servo motors require electrical signals for their operation. The ultrasonic sensor needs a trigger pulse to start their sensing operation. Similarly, the servo amplifiers in the motors need PWM signals for speed and direction control. These signals are provided by the FPGA. This is done by dividing the FPGA into modules that are dedicated for the control of the sensors and the motors. Hence there is a sensor module in the FPGA which provides the trigger pulse for the ultrasonic sensors and also computes the distance of the surrounding walls and obstacles by measuring the duration of the echo pulse output. Similarly, there is a motor driver module inside the FPGA which provides the PWM signal for both the motors. These modules are controlled by a supervisor module of the FPGA which contains the navigational algorithm for the robot. The supervisor module, motor driver module and the sensor module are coded into the FPGA hardware using the VHDL coding language. These modules are interconnected with each other by using a XILINX ISE tool. This tool provides a schematic view of these modules and facilitates their interconnection. A detailed description of these modules is carried out in the following sections.

5.1 Sensor Module

The sensor module for the FPGA controls the ultrasonic sensors on the robot platform by providing them with a trigger pulse. It also measures the duration of the echo pulse output and computes the distance of the side walls from the robot base. Since there are two ultrasonic sensors on the robot base, the sensor module is divided into two 'TRANSCIEVER' blocks, where each block controls one ultrasonic sensor. A block diagram of the 'TRANSCIEVER' block is shown in figure 5.1

Figure 5.1 A 'TRANSCIEVER' Block.

The 'TRANSCIEVER' block consists of four inputs and three outputs. The 'START' input starts the timing operation of the 'TRANSCIEVER' block. The 'RECEIVE' input is connected to the echo pulse output through appropriate interface circuitry. The 'CLK' input is used for the general timing operations of the block. The 'RESET' input is used to reset the operation of the 'TRANSCIEVER' block if it gets stuck in an unknown state. The 'TRANSMIT' output is connected to the trigger pulse input of the ultrasonic sensor. The 'COUNT' output provides the duration of the echo pulse output. The 'DONE' output acts as a signal that goes high after one sensing cycle of the 'TRANSCIEVER'.

When the 'START' input goes high, the 'TRANSCIEVER' checks the 'DONE' output. If it is in a logic '1' state the 'TRANSCIEVER' block starts its operation by pulling the 'TRANSMIT' output high for a period of 20 μ sec and changing the 'DONE' output to a logic '0'. This fires the ultrasonic sensor into the sensing cycle. The ultrasonic sensor after sending out a sonic burst pulls the echo pulse output to high. Upon sensing this transition in the 'RECEIVE' input, the 'TRANSCIEVER' block starts incrementing the 'COUNT' output. When the ultrasonic sensor receives a reflected sonic wave, it pulls down the echo pulse output. Sensing this change at the 'RECEIVE' input, the 'TRANSCIEVER' block stops incrementing the 'COUNT' output and changes the 'DONE' output signal to a high. It stays in this state until it receives a new 'START' pulse after which it starts a new sensing cycle. If the 'TRANSCIEVER' block receives a 'START' pulse and the 'DONE' output is low, it signifies that the 'TRANSCIEVER' block is in the middle of a scan cycle and its operation remains uninterrupted. A general timing diagram for the operation of 'TRANSCIEVER' block is shown in figure 5.2.

Figure 5.2 Timing for a 'TRANSCIEVER' block.

The sensor module makes use of two of these ‘TRANSCIEVER’ blocks to control the left and the right ultrasonic sensors on the robot base. The timing for these ‘TRANSCIEVER’ blocks is provided by a common clock. The start pulse, which triggers the scan cycle for the ‘TRANSCIEVER’ block is provided by a ‘TRIGGER’ block. This block provides alternate start pulses to the two ‘TRANSCIEVER’ blocks one at a time. A block diagram of the ‘TRIGGER’ block is shown in figure 5.3.

Figure 5.3 ‘TRIGGER’ block.

It has four inputs and three outputs. The ‘CLK’ input is used for general timing of the block and is tied along with the clocks of the ‘TRANSCIEVER’ blocks. The ‘Done_L’ and ‘Done_R’ inputs are connected to the ‘DONE’ outputs of the ‘TRANSCIEVER’ blocks for the left and the right sensors. The ‘RESET’ input is used to reset the operation of the ‘TRIGGER’ block. The ‘Trigger_L’ and ‘Trigger_R’ outputs are connected to the ‘START’ inputs of the two ‘TRANSCIEVER’ blocks. The ‘Data_Ready’ output provides a signal and goes high when the scan cycle for both the ‘TRANSCIEVER’ blocks is complete and the data at the ‘COUNT’ outputs is the current distance of the left and the right walls from the robot base. The ‘TRIGGER’ block starts its operation by providing a start pulse for the left sensor by pulling up the ‘Trigger_L’ output. It then waits for the ‘TRANSCIEVER’ block of the left sensor to complete its scan cycle. When the

‘Done_L’ input goes high the ‘TRIGGER’ block pulls the ‘Trigger_R’ output and starts the scan cycle for the right sensor. When the ‘Done_R’ input goes high the ‘TRIGGER’ block understands that the scan cycles for both the ‘TRANSCIEVER’ blocks are complete and it pulls the ‘Data_Ready’ output high. The ‘TRIGGER’ block stays in this state for a period of 10 msec after which it triggers the scan cycle for the left sensor again.

The data at the ‘COUNT’ output for the two ‘TRANSCIEVER’ blocks remains stable until the start of a new scan cycle. During the duration of the scan cycle, the data at the ‘COUNT’ output keeps incrementing and is not a correct indication of the distance of the walls from the robot base. If this data is read during the scan cycle, it will cause errors in the robot navigation. In order to avoid this, a ‘LATCH’ block is used. This block captures the data at the ‘COUNT’ output when the data is stable. A block diagram of the ‘LATCH’ block is shown in figure 5.4.

Figure 5.4 ‘LATCH’ block.

The ‘Data_in’ input is connected to the ‘COUNT’ output of the ‘TRANSCIEVER’ block. The ‘HOLD’ input is connected to the ‘Data_Ready’ output of the ‘TRIGGER’ block. The ‘CLK’ input is tied together with the clocks for the other two blocks. The ‘Data_out’

output is the current value of distance of the left and the right walls. A block diagram, showing the interconnection of all the blocks is shown in figure 5.5.

Figure 5.5 Interconnection of all the blocks.

The 'TRANSCIEVER' blocks have a 'RESET' input which when pulled high interrupt their operation, 'COUNT' outputs set to zero and bring them back to a preset stage where they wait for the start pulse. This avoids the incorrect operation of the 'TRANSCIEVER' blocks if they enter an unknown state. A block diagram of the 'RESET' block is shown in figure 5.6.

Figure 5.6 'RESET' block.

The 'CLK' input is tied to the clocks for the other blocks. The 'Reset_in' input is connected to the 'DONE' output of the 'TRANSCIEVER' block. The 'Reset_out' output is connected to the 'RESET' input of the 'TRANSCIEVER' block. When the 'Reset_in' input or the 'DONE' output of the 'TRANSCIEVER' block goes low, the 'RESET' block starts counting the duration of this low pulse. The period for which the 'DONE' output remains low is the duration of the scan cycle for the ultrasonic sensor. This is approximately equivalent to 30 msec. If the 'DONE' output does not go high after this time span, it indicates that the 'TRANSCIEVER' block has entered in an unknown state and is not responding and this block needs to be reset. Hence the 'RESET' block counts up to 30 msec and if the 'DONE' output does not go high, it resets the 'TRANSCIEVER' block by pulling up the 'Reset_out' output. The block diagram of the sensor module

containing the ‘TRANSCIEVER’, ‘TRIGGER’ and the ‘RESET’ blocks is shown in figure 5.7.

Figure 5.7 Block diagram of the Sensor Module.

5.2 Motor Driver Module

The motor driver module for the FPGA controls the speed and the direction of the two servo motors by providing them with a PWM input. A block diagram of motor driver module is shown in figure 5.8.

Figure 5.8 Block diagram for motor driver module.

The 'CLK' input is connected to the FPGA clock source pin. The 'PWM_L' and 'PWM_R' inputs are the speed inputs for the left and the right motors. These inputs are supplied by the supervisor module in accordance with the required turn angle for the robot. The 'OBS' input is connected to the output of the front infrared sensor of the robot platform. The 'Data_L' and 'Data_R' inputs are connected to the latch outputs of the supervisor module. These inputs are used to determine the direction for turning the robot if the 'OBS' goes high. The 'Switch' input, connected to the output of a DIP switch on the FPGA board, when high provides the PWM input for the two servo motors. When this input is low the PWM outputs from the motor driver module are disabled and the

motors are stationary. This input is used to start the physical movement of the robot by applying the PWM to the left and right servo motors and was used during the testing of the navigation algorithm for the robot. The 'PWM' and 'PWM1' outputs are the pulse width modulation outputs for the left and the right motors. The 'DISP' output is connected to an LED on the FPGA platform and is used to indicate the presence of an obstacle in front of the robot. The LED starts to glow if there is an obstacle present and is turned off if the path in front of the robot is clear.

5.3 Supervisor Module

The supervisor module for the FPGA acts as the decision making component for the robot navigation. It takes in the data from the left and the right sensors and decides the turn angle for the robot. This turn angle is converted into speeds for the left and the right motors. These speeds are passed onto the motor driver module which converts them into PWM outputs for the left and right servo motors. It is the supervisor module that holds the layered motion controller. The decisions regarding the turn angle for the robot are made by this layered motion controller. A block diagram for the supervisor module is shown in figure 5.9

Figure 5.9 Block diagram of the supervisor module.

The 'CLK' input is connected to the FPGA clock source pin. The 'DATA_L' and 'DATA_R' inputs are connected to the latch outputs of the sensor module. The 'DONE' input is connected to the 'Data_Ready' output of the sensor module. When this input goes high, the supervisor module latches onto the data for the left and the right sensors. This data is used layered motion controller for determining the turn angle for the robot. This turn angle is converted into speeds for the left and the right motors. The 'PWM_L'

and 'PWM_R' are the speeds for the left and the right motors. These are connected to the PWM inputs for the motor driver module. The 'EDOT' and 'Data' outputs are used for observation purposes and represent the rate of change of error and the error for each scan cycle of the left and right sensors. These outputs were used during the testing of the navigational algorithm for the robot and are viewable on the LCD screen of the FPGA development board. The LCD of the FPGA requires an interface block to view this data on the screen. This interface block is capable of displaying four different sets of 22 bit data. The interface block is shown in figure 5.10.

Figure 5.10 Interface block for the LCD screen.

The 'CLK' input is connected to the FPGA clock source pin. The 'DATA_L', 'DATA_R', 'DATA_L1', and 'DATA_R1' inputs are the four sets of 22 bit data to be displayed on the LCD screen. The 'Trigger' input is used to refresh the data on the LCD screen and is connected to the 'Data_Ready' output of the sensor module. All the outputs of the interface module are connected to the input pins of the LCD screen and provide various control and data signals for displaying the data on the screen. The data0-data7 outputs provide data for the data register and the instruction register. The 'RS' input when high indicates the data for the data register and when low indicates the data for the instruction register. The 'EN' output enables the display on the LCD screen.. A block diagram showing the interconnection of the sensor module, supervisor module and the interface block is shown in figure 5.11.

Figure 5.11 Block diagram showing the interconnection of various modules.

Figure 5.12 Block diagram of overall navigational controller for the robot.

The schematic in figure 5.12 provides an overall navigational control for the robot and includes all of its modules. The VHDL code for these modules is included in the appendix.

5.4 Summary

This chapter discussed the software involved in the realization of effective robot navigation. The FPGA code was divided into different modules to control the respective hardware modules on the robot. The sensor module of the robot was used to control all the sensors, the motor driver module provided a control for the servo motors, and the supervisor module used the multilayered navigational algorithm for making the decisions regarding the turn angle for the robot. The next chapter provides the results of implementation of the multilayered navigational control algorithm on the robot system.

Chapter 6

Results

This thesis aims at the navigational control of an autonomous mobile robot through passages and hallways. The navigation strategy for the robot varies with the type of the passage. If the robot is moving through a passage where the side sensors detect the walls on both sides, the robot tries to move in the center. For passages where one of the walls ceases to exist or is beyond the range of the side sensor, the robot moves parallel to the detected wall at a pre-defined distance value. While navigating through a passage if the front sensor of the robot detects an object within a preset range, the navigational algorithm is intelligent enough to move around the obstacle and avoid any collisions. These tasks for robot navigation are achieved by employing a control strategy in which a fully developed controller is built from elementary layers of control. The outputs of the individual layers are integrated into a single control action. This involves the utilization of three layers of control including centering layer, wall hugging layer, and obstacle avoidance layer. All but obstacle avoidance layer are developed using fuzzy proportional plus derivative controller. These layers are implemented on an FPGA by using the VHDL coding language and are a part of the supervisor module of FPGA software code. It is this module which acts as the brain of the robot and makes all the decisions regarding the navigation strategy.

When this module is implemented on an FPGA and the robot is allowed to start its navigation from one corner of the passage with walls on both sides, it starts to move towards the center and reaches the center of the passage in typically four to five seconds. It then continues to move in the center for the remaining length of the hallway. However if it senses an obstacle while moving in the center or while trying to achieve the center of the passage, the obstacle avoidance layer is activated and the forward motion of the robot is discontinued. The robot then starts moving backwards to avoid getting too close to the obstacle and makes a turn of 45 degrees in the direction indicated by the side sensors. If the left sensor value is greater than the right sensor value the robot turns towards its left and does the opposite if the right sensor value is greater than the left sensor value. This allows the robot to steer past the obstacle safely. A schematic of the robot environment is shown in figure 6.1.

Figure 6.1 Schematic of robot environment.

Now since there is only one wall to the left the robot moves parallel to the sensed wall at a preset distance value. If the robot is too close to the wall then it starts moving away from the wall until it reaches the predefined distance and if it is too far away it moves

towards the wall. The obstacle avoidance works in a similar fashion if the robot detects an object while in the wall hugging mode. Also during its motion in the wall hugging mode if the robot detects both the side walls, the centering layer takes control over the navigation strategy for the robot. The navigational algorithm makes similar transition to the wall hugging mode from the centering mode if a situation arises.

The robot navigation strategy is captured as a video clip (CD attached with this thesis) where the robot is shown to be traveling in the centering mode, wall hugging mode, and obstacle avoidance mode. It also shows the transitions from centering to obstacle avoidance, wall hugging to obstacle avoidance, centering to wall hugging, and vice-versa.

Appendix-VHDL Programs

1. Transceiver Block

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

entity TRANSCIEVER is
    port(START, RECIEVE: in STD_LOGIC;
         CLK: in STD_LOGIC;
         RESET: in STD_LOGIC;
         TRANSMIT: out STD_LOGIC;
         DONE: out STD_LOGIC;
         COUNT: out STD_LOGIC_VECTOR(21 downto 0));
end TRANSCIEVER;

architecture Behavioral of Transciever is

    type STATE_TYPE is (M_Ready, M_Start, M_Start1, M_Transmit, M_Wait1,
        M_wait2, M_End, M_Reset);
    signal CS, NS: STATE_TYPE;
    signal counter: STD_LOGIC_VECTOR(21 downto 0);
    signal flag_count: STD_LOGIC;

begin
    COUNT<=counter;
    SYNC_PROC: process (CLK, RESET)
    begin
        if (RESET = '1') then
            CS<=M_Reset;
        elsif (CLK'event and CLK = '1') then
            if(CS=M_Start) then
                counter<=(others=>'0');
            elsif (flag_count='1') then
                counter<=counter+1;
            end if;
            CS <= NS;
        end if;
    end process;

    COMB_PROC: process (CS, START, RECIEVE, counter)
    begin
        case CS is
            when M_Ready =>
                flag_count<='0';
                DONE<='1';
```

```

--      TRANSMIT<='0';
--      COUNT<=counter;
--      if ( START='1' ) then
--          NS<=M_Start;
--      else
--          NS<=M_Ready;
--      end if;

when M_Start =>
    flag_count<='0';
    DONE<='0';
    TRANSMIT<='0';
    NS<=M_Start1;

when M_Start1 =>
    flag_count<='1';
    DONE<='0';
    TRANSMIT<='0';
--    COUNT<=counter;
--    if ( counter="000000000000000000000000" ) then
--        NS<=M_Transmit;
--    else
--        NS<=M_Start1;
--    end if;

when M_Transmit =>
    flag_count<='1';
    DONE<='0';
    TRANSMIT<='1';
--    COUNT<=counter;
--    if ( counter="000010000000000000000000" ) then
--        NS<=M_Wait1;
--    else
--        NS<=M_Transmit;
--    end if;

when M_Wait1 =>
    flag_count<='0';
    DONE<='0';
    TRANSMIT<='0';
--    COUNT<=counter;
--    if ( RECIEVE='1' ) then
--        NS<=M_Wait2;
--    else
--        NS<=M_Wait1;
--    end if;

when M_Wait2 =>
    flag_count<='1';
    DONE<='0';

```

```

--      TRANSMIT<='0';
--      COUNT<=counter;
--      if (RECIEVE='0') then
--      NS<=M_End;
--      else
--      NS<=M_Wait2;
--      end if;

--      when M_End =>
--      flag_count<='0';
--      DONE<='0';
--      TRANSMIT<='0';
--      COUNT<=counter;
--      NS<=M_Ready;

--      when M_Reset =>
--      flag_count<='0';
--      DONE<='0';
--      TRANSMIT<='0';
--      NS<=M_Ready;

--      end case;
--      end process;

--      end Behavioral;

```

2. Trigger Block

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Trigger4 is
    Port(
        CLK: in STD_LOGIC;
        Done_L: in STD_LOGIC;
        Done_R: in STD_LOGIC;
        Reset: in STD_LOGIC;
        Data_Ready: out STD_LOGIC;
        Trigger_L: out STD_LOGIC;
        Trigger_R: out STD_LOGIC);
end Trigger4 ;

architecture Behavioral of Trigger4 is

    type STATE_TYPE is (Initialize, L_Trigger, L_Wait, R_Trigger,
        R_Wait, Complete);
    signal CS, NS: STATE_TYPE;
    signal flag_count_L:STD_LOGIC;
    signal flag_count_R:STD_LOGIC;
    signal count_L:STD_LOGIC_VECTOR(24 downto 0);
    signal count_R:STD_LOGIC_VECTOR(24 downto 0);

begin
    process(CLK)
    begin
        if(Reset='1') then
            CS<=Initialize;
        elsif (CLK'event and CLK = '1') then
            if(CS=Initialize) then
                count_L<=(others=>'0');
                count_R<=(others=>'0');
            elsif (flag_count_L='1') then
                count_L<=count_L+1;
            elsif (flag_count_R='1') then
                count_R<=count_R+1;
            end if;
            CS <= NS;
        end if;
    end process;

    COMB_PROC: process (CS, Done_L, count_L, Done_R, count_R)
    begin
```

```

case CS is
  when Initialize=>
    flag_count_L<='0';
    flag_count_R<='0';
    Data_Ready<='0';
    Trigger_L<='0';
    Trigger_R<='0';
    NS<=L_Trigger;

  when L_Trigger =>
    flag_count_L<='1';
    flag_count_R<='0';
    Data_Ready<='0';
    Trigger_L<='1';
    Trigger_R<='0';
    if(count_L>="000000000010000000000") then
      NS<=L_Wait;
    else
      NS<=L_Trigger;
    end if;

  when L_Wait =>
    flag_count_L<='0';
    flag_count_R<='0';
    Data_Ready<='0';
    Trigger_L<='0';
    Trigger_R<='0';
    if (Done_L='1') then
      NS<=R_Trigger;
      NS<=Complete;
    else
      NS<=L_Wait;
    end if;

  when R_Trigger =>
    flag_count_L<='0';
    flag_count_R<='1';
    Data_Ready<='0';
    Trigger_L<='0';
    Trigger_R<='1';
    if(count_R>="000000000010000000000") then
      NS<=R_Wait;
    else
      NS<=R_Trigger;
    end if;

  when R_Wait =>
    flag_count_L<='0';
    flag_count_R<='0';
    Data_Ready<='0';

```

```

        Trigger_L<='0';
        Trigger_R<='0';
    if (Done_R='1') then
        NS<=Complete;
    else
        NS<=R_Wait;
    end if;

    when Complete=>
        flag_count_L<='1';
        flag_count_R<='1';
        Data_Ready<='1';
        Trigger_L<='0';
        Trigger_R<='0';
        if ( count_L="11111111111111111111111111111111") then
            NS<=Initialize;
        else
            NS<=Complete;
        end if;

    end case;
end process;

end Behavioral;

```


3. Latch Block

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Latch is
    Port(
        CLK: in STD_LOGIC;
        Hold: in STD_LOGIC;
        Data_in: in STD_LOGIC_VECTOR(21 downto 0);
        Data_out: out STD_LOGIC_VECTOR(21 downto 0));
end Latch;

architecture Behavioral of Latch is

    type STATE_TYPE is (Initialize, Low, High);
    signal CS, NS: STATE_TYPE;
    signal Data: STD_LOGIC_VECTOR(21 downto 0);

begin
    Data_out<=Data;
    process(CLK)
    begin
        if (CLK'event and CLK = '1') then
            CS <= NS;
        end if;
    end process;

    COMB_PROC: process (CS, Hold)
    begin
        case CS is
            when Initialize=>
                Data<=(others=>'0');
                NS<=Low;

            when Low =>
                if(Hold='1') then
                    NS<=High;
                else
                    NS<=Low;
                end if;

            when High=>
                Data(21 downto 0)<=Data_in(21 downto 0);
        end case;
    end process;
end Behavioral;
```

```

--
    if (Hold='0') then
    NS<=Low;
    NS<=Complete;
    else
    NS<=High;
    end if;

    end case;
end process;

end Behavioral;
```

4. Reset Block

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Reset_Logic is
    Port(
        CLK: in STD_LOGIC;
        Reset_in: in STD_LOGIC;
        Reset_out: out STD_LOGIC);
end Reset_Logic;

architecture Behavioral of Reset_Logic is
    signal count:STD_LOGIC_VECTOR(23 downto 0);

begin
    process(CLK)
    begin
        if (CLK='1' and CLK'event) then
            if(Reset_in='0') then
                count<=count+1;
                if(count<="001111111111111111111111") then
                    Reset_out<='0';
                elsif(count>="001111111111111111111111") then
                    Reset_out<='1';
                elsif(count>="011111111111111111111111") then
                    Reset_out<='0';
                end if;
            else
                count<=(others=>'0');
                Reset_out<='0';
            end if;
        end if;
    end process;

end Behavioral;
```

5. Motor Driver Module

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Motor_Ctrl3 is
    Port(
        CLK: in STD_LOGIC;
        Switch: in STD_LOGIC;
        Obs: in STD_LOGIC;
        Data_L: in STD_LOGIC_VECTOR(21 downto 0);
        Data_R: in STD_LOGIC_VECTOR(21 downto 0);
        PWM_L: in STD_LOGIC_VECTOR(21 downto 0);
        PWM_R: in STD_LOGIC_VECTOR(21 downto 0);
        Disp: out STD_LOGIC;
        PWM: out STD_LOGIC;
        PWM1: out STD_LOGIC);
end Motor_Ctrl3;

architecture Behavioral of Motor_Ctrl3 is
    signal count_m:STD_LOGIC_VECTOR(24 downto 0);
    signal count:STD_LOGIC_VECTOR(21 downto 0);
    signal count1:STD_LOGIC_VECTOR(21 downto 0);
    signal Left_high:STD_LOGIC_VECTOR(21 downto 0);
    signal Right_high:STD_LOGIC_VECTOR(21 downto 0);
    signal Detected:STD_LOGIC;
    signal Obs_Count:STD_LOGIC_VECTOR(2 downto 0);

begin
    Disp<=not Obs;
    process(CLK, Obs, Switch)
    -- variable count:STD_LOGIC_VECTOR(21 downto 0);
    begin
        if (CLK='1' and CLK'event) then
            if(Switch='0') then
                If(Obs='1') then
                    Detected<='1';
                    count_m<=(others=>'0');
                    Obs_Count<=(others=>'0');
                end if;
                If(Detected='1') then
                    count_m<=count_m+1;
                    if(count_m="111111111111111111111111") then
                        Obs_Count<=Obs_Count+1;
                    end if;
                    if(Obs_Count<"100") then
                        Left_high<="0000100010000100111111";
                    end if;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

```

        Right_high<="0000100111011011001101";
    elseif(Obs_Count="100") then
        if(Data_R>=Data_L) then

Left_high<="0000101010100011100001";

Right_high<="0000100100100111110000";
            else

Left_high<="0000100100110010100110";

Right_high<="0000011111010111011000";
                end if;
            elseif(Obs_Count="111") then
                Left_high<="0000101010001101110101";
                Right_high<="0000100000000000001011";
                Detected<='0';
            end if;
        else
            Left_high<=PWM_L;
            Right_high<=PWM_R;
        end if;
        count<=count+1;
        count1<=count1+1;
        if(count>"1000000111001000110101") then
            count<=(others=>'0');
            PWM<='1';
        elseif(count>Left_high) then
            PWM<='0';
        end if;
        if(count1>"1000000111001000110101") then
            count1<=(others=>'0');
            PWM1<='1';
        elseif(count1>Right_high) then
            PWM1<='0';
        end if;
    end if;
end if;
end process;

end Behavioral;

```

6. Supervisor Module

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

----MODIFIED RULE BASE FOR THE WHOLE AND ENCODERS REMOVED
entity Controller_PD9 is
    Port(
        CLK: in STD_LOGIC;
        Reset: in STD_LOGIC;
        Done: in STD_LOGIC;
        Data_L: in STD_LOGIC_VECTOR(21 downto 0);
        Data_R: in STD_LOGIC_VECTOR(21 downto 0);
        PWM_L: out STD_LOGIC_VECTOR(21 downto 0);
        Edot: out STD_LOGIC_VECTOR(21 downto 0);
        Data: out STD_LOGIC_VECTOR(21 downto 0);
        PWM_R: out STD_LOGIC_VECTOR(21 downto 0));
end Controller_PD9;

architecture Behavioral of Controller_PD9 is
    signal Err:STD_LOGIC_VECTOR(21 downto 0);
    -- signal Edot:STD_LOGIC_VECTOR(21 downto 0);
begin
    process(CLK, Done)
        variable RB_L :STD_LOGIC_VECTOR(21 downto 0):="0000101010001101110101";--02A375
        variable RB_R :STD_LOGIC_VECTOR(21 downto 0):="0000100011010000111100";--02343C
        variable RMB_L:STD_LOGIC_VECTOR(21 downto 0):="0000101010001101110101";--02A375
        variable RMB_R:STD_LOGIC_VECTOR(21 downto 0):="0000100010110000011001";--022C19
        variable RMS_L:STD_LOGIC_VECTOR(21 downto 0):="0000101010001101110101";--02A375
        variable RMS_R:STD_LOGIC_VECTOR(21 downto 0):="0000100010001111110110";--0223F6
        variable RS_L :STD_LOGIC_VECTOR(21 downto 0):="0000101010001101110101";--02A375
        variable RS_R :STD_LOGIC_VECTOR(21 downto 0):="0000100001111010001001";--021E89
        variable ZO_L :STD_LOGIC_VECTOR(21 downto 0):="0000101010001101110101";--02A375
        variable ZO_R :STD_LOGIC_VECTOR(21 downto 0):="0000011111010111010000";--01F5D0
```

```

        variable LS_L :STD_LOGIC_VECTOR(21 downto
0):="0000100111101011000100";--027AC4
        variable LS_R :STD_LOGIC_VECTOR(21 downto
0):="0000011111010111010000";--01F5D0
        variable LMS_L:STD_LOGIC_VECTOR(21 downto
0):="0000100111010101010111";--027557
        variable LMS_R:STD_LOGIC_VECTOR(21 downto
0):="0000011111010111010000";--01F5D0
        variable LMB_L:STD_LOGIC_VECTOR(21 downto
0):="0000100110110100110100";--026D34
        variable LMB_R:STD_LOGIC_VECTOR(21 downto
0):="0000011111010111010000";--01F5D0
        variable LB_L :STD_LOGIC_VECTOR(21 downto
0):="0000100110010100010001";--026511
        variable LB_R :STD_LOGIC_VECTOR(21 downto
0):="0000011111010111010000";--01F5D0
        variable Error:STD_LOGIC_VECTOR(21 downto
0):="0000000000000000000000";
        variable Error_i:STD_LOGIC_VECTOR(21 downto
0):="0000000000000000000000";
        variable Edot_i:STD_LOGIC_VECTOR(21 downto
0):="0000000000000000000000";
        variable Last_Error:STD_LOGIC_VECTOR(21 downto
0):="0000000000000000000000";
        variable value_L:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable value_R:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable center1:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable center2:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable center3:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable center4:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable error_element1:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable error_element2:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable error_element3:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable error_element4:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable edot_element1:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable edot_element2:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable edot_element3:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable edot_element4:STD_LOGIC_VECTOR(22 downto
0):="0000000000000000000000";
        variable Error_sign:STD_LOGIC:='0';
        variable Last_Error_sign:STD_LOGIC:='0';
        variable flag_reset:STD_LOGIC_VECTOR(2 downto 0):="000";
        variable Edot_sign:STD_LOGIC:='0';
        variable error_region:STD_LOGIC_VECTOR(1 downto 0);

```

```

variable edot_region:STD_LOGIC_VECTOR(1 downto 0);
variable error_mem1:STD_LOGIC_VECTOR(3 downto 0);
variable error_mem2:STD_LOGIC_VECTOR(3 downto 0);
variable edot_mem1:STD_LOGIC_VECTOR(3 downto 0);
variable edot_mem2:STD_LOGIC_VECTOR(3 downto 0);
begin
    if(Reset='1') then
        Last_Error:=Error;
        flag_reset:="111";
    if(CLK='1' and CLK'Event) then
        if(Done='1') then
            if(flag_reset="110") then
                flag_reset:="101";
            end if;
            if(flag_reset="100") then
                flag_reset:="011";
            end if;
            if(flag_reset="010") then
                flag_reset:="001";
            end if;
            if(flag_reset="00") then
                if(Data_L>=Data_R) then
                    Error:=Data_L-Data_R;
                    Error_sign:='0';
                else
                    Error:=Data_R-Data_L;
                    Error_sign:='1';
                end if;
                Err<=Error;
                if(Error_sign=Last_Error_sign) then
                    if(Error_sign='0') then
                        if(Last_Error>=Error) then
                            Edot_i:=Last_Error-Error;
                            Edot_sign:='1';
                        else
                            Edot_i:=Error-Last_Error;
                            Edot_sign:='0';
                        end if;
                    else
                        if(Last_Error>=Error) then
                            Edot_i:=Last_Error-Error;
                            Edot_sign:='0';
                        else
                            Edot_i:=Error-Last_Error;
                            Edot_sign:='1';
                        end if;
                    end if;
                else
                    Edot_i:=Error+Last_Error;
                    Edot_sign:=Error_sign;
                end if;
            if(Edot_i>"0000001010000000000000") then
                Edot_i:="0000000000000000000000";
            end if;
            Edot<=Edot_i;
            value_L(21 downto 0):=Data_L;
            value_L(22):='0';

```



```

value_R(21 downto 0):=Data_R;
value_R(22):='0';
center4:=value_L+value_R;
center1(20 downto 0):=center4(22 downto 2);
center1(22 downto 21):=(others=>'0');
center2(21 downto 0):=center4(22 downto 1);
center2(22):='0';
center3:=center1+center2;
error_element1(18 downto 0):=center1(22 downto
4);
error_element2(19 downto 0):=center1(22 downto
3);
error_element3(20 downto 0):=center1(22 downto
2);
error_element4(21 downto 0):=center1(22 downto
1);

Error_i:=Error;
if(Error<=center1) then
    error_region:="00";
elsif(Error>center1 and Error<=center2) then
    error_region:="01";
    Error:=Error-center1;
elsif(Error>center2 and Error<=center3) then
    error_region:="10";
    Error:=Error-center2;
elsif(Error>center3 and Error<=center4) then
    error_region:="11";
    Error:=Error-center3;
end if;
if(error>=error_element1) then
    if(error>=error_element2) then

if(error>=(error_element2+error_element1)) then
    if(error>=(error_element3))
then
        if(error>=(error_element3+error_element1)) then

if(error>=(error_element3+error_element2)) then

if(error>=(error_element3+error_element2+error_element1)) then

if(error>=(error_element4)) then

if(error>=(error_element4+error_element1)) then

if(error>=(error_element4+error_element2)) then

        if(error>=(error_element4+error_element2+error_element1))
then
            if(error>=(error_element4+error_element3)) then

if(error>=(error_element4+error_element3+error_element1)) then

```

```

        if(error>=(error_element4+error_element3+error_element2)) then

            if(error>=(error_element4+error_element3+error_element2+error_ele
ment1)) then

                error_mem2:="1111";

            else

                error_mem2:="1110";

            end if;

        else

            error_mem2:="1101";

        end if;

    else

        error_mem2:="1100";

    end if;

else

    error_mem2:="1011";

end if;

else

    error_mem2:="1010";

end if;

else

    error_mem2:="1001";

end if;

else

    error_mem2:="1000";

end if;

else

    error_mem2:="0111";

end

if;

else

```

```

error_mem2:="0110";
end if;
else

error_mem2:="0101";
end if;
else

error_mem2:="0100";
end if;
else
error_mem2:="0011";
end if;
else
error_mem2:="0010";
end if;
else
error_mem2:="0001";
end if;
else
error_mem2:="0000";
end if;
error_mem1:=(("1111")-error_mem2);
center1:="000000000011000000000000";
center2:="000000000110000000000000";
center3:="000000001010000000000000";
center4:="000000001110000000000000";
--
center1:="000000000100000000000000";
--
center2:="000000001000000000000000";
--
center3:="000000001100000000000000";
--
center4:="000000010000000000000000";
edot_element1(18 downto 0):=center1(22 downto
4);
edot_element2(19 downto 0):=center1(22 downto
3);
edot_element3(20 downto 0):=center1(22 downto
2);
edot_element4(21 downto 0):=center1(22 downto
1);

if(Edot_i<=center1) then
edot_region:="00";
elsif(Edot_i>center1 and Edot_i<=center2) then
edot_region:="01";
Edot_i:=Edot_i-center1;
elsif(Edot_i>center2 and Edot_i<=center3) then
edot_region:="10";
Edot_i:=Edot_i-center2;
elsif(Edot_i>center3 and Edot_i<=center4) then
edot_region:="11";
Edot_i:=Edot_i-center3;
end if;
if(error>=edot_element1) then
if(error>=edot_element2) then

if(error>=(edot_element2+edot_element1)) then

```

```

                                if(error>=(edot_element3))
then
    if(error>=(edot_element3+edot_element1)) then
    if(error>=(edot_element3+edot_element2)) then
    if(error>=(edot_element3+edot_element2+edot_element1)) then
    if(error>=(edot_element4)) then
    if(error>=(edot_element4+edot_element1)) then
    if(error>=(edot_element4+edot_element2)) then
        if(error>=(edot_element4+edot_element2+edot_element1)) then
            if(error>=(edot_element4+edot_element3)) then

    if(error>=(edot_element4+edot_element3+edot_element1)) then

    if(error>=(edot_element4+edot_element3+edot_element2)) then

        if(error>=(edot_element4+edot_element3+edot_element2+edot_element
1)) then

                                edot_mem2:="1111";
                                else
                                edot_mem2:="1110";
                                end if;
                                else
                                edot_mem2:="1101";
                                end if;
                                else
                                edot_mem2:="1100";
                                end if;
                                else
                                edot_mem2:="1011";
                                end if;
                                else

```

```

        edot_mem2:="1010";

    end if;

else

    edot_mem2:="1001";

end if;

else

    edot_mem2:="1000";

end if;

else

    edot_mem2:="0111";

end if;

if;

else

    edot_mem2:="0110";

end if;

else

    edot_mem2:="0101";

end if;

else

    edot_mem2:="0100";

end if;

else

    edot_mem2:="0011";

end if;

else

    edot_mem2:="0010";

end if;

else

    edot_mem2:="0001";

end if;

else

    edot_mem2:="0000";

end if;

edot_mem1:=(("1111")-edot_mem2);
Data<=Err;
Data(21 downto 20)<= error_region(1 downto 0);
Data(19 downto 16)<=error_mem1(3 downto 0);
Data(15 downto 12)<=error_mem2(3 downto 0);
Data(11 downto 10)<=(others=>'0');
Data(9 downto 8)<=edot_region;
Data(7 downto 4)<=edot_mem1;
Data(3 downto 0)<=edot_mem2;

if(error_region="00")then

```

```

-----ZERO AND PS FOR ERROR-----
-----
if(error_sign='0') then
    if(error_mem1>="1010") then-----
--ZERO EROR-----
    if(edot_region="00") then----
---ZERO EDOT-----
        PWM_L<=ZO_L;
        PWM_R<=ZO_R;
    elsif(edot_region="01") then
        if(Edot_sign='0') then-
-----PS EDOT-----
            PWM_L<=LS_L;
            PWM_R<=LS_R;
        else-----
-----NS EDOT-----
            PWM_L<=RS_L;
            PWM_R<=RS_R;
        end if;
    elsif(edot_region="10") then
        if(Edot_sign='0') then-
-----PMS EDOT-----
            PWM_L<=LS_L;
            PWM_R<=LS_R;
        else-----
-----NMS EDOT-----
            PWM_L<=RS_L;
            PWM_R<=RS_R;
        end if;
    elsif(edot_region="11") then
        if(Edot_sign='0') then-
-----PMB EDOT-----
            PWM_L<=LS_L;
            PWM_R<=LS_R;
        else-----
-----NMB EDOT-----
            PWM_L<=RS_L;
            PWM_R<=RS_R;
        end if;
    end if;
else-----
--PS ERROR-----
    if(edot_region="00") then----
---ZERO EDOT-----
        PWM_L<=LS_L;
        PWM_R<=LS_R;
    elsif(edot_region="01") then
        if(Edot_sign='0') then-
-----PS EDOT-----
            PWM_L<=LMS_L;
            PWM_R<=LMS_R;
        else-----
-----NS EDOT-----
            PWM_L<=ZO_L;
            PWM_R<=ZO_R;
        end if;
    elsif(edot_region="10") then

```



```

                                PWM_L<=RS_L;
                                PWM_R<=RS_R;
                                end if;
                                end if;
else-----
--NS ERROR-----
--ZERO EDOT-----
                                if(edot_region="00") then----
                                    PWM_L<=RS_L;
                                    PWM_R<=RS_R;
                                elsif(edot_region="01") then
                                    if(Edot_sign='0') then-
-----PS EDOT-----
                                    PWM_L<=ZO_L;
                                    PWM_R<=ZO_R;
                                else-----
-----NS EDOT-----
                                    PWM_L<=RMS_L;
                                    PWM_R<=RMS_R;
                                end if;
                                elsif(edot_region="10") then
                                    if(Edot_sign='0') then-
-----PMS EDOT-----
                                    PWM_L<=LS_L;
                                    PWM_R<=LS_R;
                                else-----
-----NMS EDOT-----
                                    PWM_L<=RMS_L;
                                    PWM_R<=RMS_R;
                                end if;
                                elsif(edot_region="11") then
                                    if(Edot_sign='0') then-
-----PMB EDOT-----
                                    PWM_L<=LMS_L;
                                    PWM_R<=LMS_R;
                                else-----
-----NMB EDOT-----
                                    PWM_L<=RMS_L;
                                    PWM_R<=RMS_R;
                                end if;
                                end if;
                                end if;
                                end if;

                                elsif(error_region="01") then
-----PS AND PMS FOR ERROR-----
-----
                                if(error_sign='0') then
                                    if(error_mem1>="0111") then-----
-----PS ERROR-----
                                    if(edot_region="00") then----
-----ZERO EDOT-----
                                        PWM_L<=LS_L;
                                        PWM_R<=LS_R;

```



```

-----PS EDOT-----
-----NS EDOT-----
-----PMS EDOT-----
-----NMS EDOT-----
-----PMB EDOT-----
-----NMB EDOT-----
---PMS ERROR-----
---ZERO EDOT-----
-----PS EDOT-----
-----NS EDOT-----
-----PMS EDOT-----
-----NMS EDOT-----

elseif(edot_region="01") then
    if(Edot_sign='0') then-
        PWM_L<=LMS_L;
        PWM_R<=LMS_R;
    else-----
        PWM_L<=ZO_L;
        PWM_R<=ZO_R;
    end if;
elseif(edot_region="10") then
    if(Edot_sign='0') then-
        PWM_L<=LMS_L;
        PWM_R<=LMS_R;
    else-----
        PWM_L<=RS_L;
        PWM_R<=RS_R;
    end if;
elseif(edot_region="11") then
    if(Edot_sign='0') then-
        PWM_L<=LMS_L;
        PWM_R<=LMS_R;
    else-----
        PWM_L<=RMS_L;
        PWM_R<=RMS_R;
    end if;
end if;
else-----
    if(edot_region="00") then----
        PWM_L<=LMS_L;
        PWM_R<=LMS_R;
    elseif(edot_region="01") then
        if(Edot_sign='0') then-
            PWM_L<=LMS_L;
            PWM_R<=LMS_R;
        else-----
            PWM_L<=ZO_L;
            PWM_R<=ZO_R;
        end if;
    elseif(edot_region="10") then
        if(Edot_sign='0') then-
            PWM_L<=LMS_L;
            PWM_R<=LMS_R;
        else-----
            PWM_L<=RS_L;
            PWM_R<=RS_R;
        end if;
    end if;
end if;

```

```

elseif(edot_region="11") then
    if(Edot_sign='0') then-
-----PMB EDOT-----
        PWM_L<=LMS_L;
        PWM_R<=LMS_R;
    else-----
-----NMB EDOT-----
        PWM_L<=RS_L;
        PWM_R<=RS_R;
    end if;
end if;
-----NS AND NMS FOR ERROR-----
-----
else
    if(error_mem1>="0111") then-----
--NS ERROR-----
        if(edot_region="00") then----
---ZERO EDOT-----
            PWM_L<=RS_L;
            PWM_R<=RS_R;
        elseif(edot_region="01") then
            if(Edot_sign='0') then-
-----PS EDOT-----
                PWM_L<=ZO_L;
                PWM_R<=ZO_R;
            else-----
-----NS EDOT-----
                PWM_L<=RMS_L;
                PWM_R<=RMS_R;
            end if;
        elseif(edot_region="10") then
            if(Edot_sign='0') then-
-----PMS EDOT-----
                PWM_L<=LS_L;
                PWM_R<=LS_R;
            else-----
-----NMS EDOT-----
                PWM_L<=RMS_L;
                PWM_R<=RMS_R;
            end if;
        elseif(edot_region="11") then
            if(Edot_sign='0') then-
-----PMB EDOT-----
                PWM_L<=LMS_L;
                PWM_R<=LMS_R;
            else-----
-----NMB EDOT-----
                PWM_L<=RMS_L;
                PWM_R<=RMS_R;
            end if;
        end if;
    else-----
--NMS ERROR-----
        if(edot_region="00") then----
---ZERO EDOT-----
            PWM_L<=RMS_L;

```

```

PWM_R<=RMS_R;
elseif(edot_region="01") then
    if(Edot_sign='0') then-
-----PS EDOT-----
        PWM_L<=ZO_L;
        PWM_R<=ZO_R;
    else-----
        PWM_L<=RMS_L;
        PWM_R<=RMS_R;
    end if;
elseif(edot_region="10") then
    if(Edot_sign='0') then-
-----PMS EDOT-----
        PWM_L<=LS_L;
        PWM_R<=LS_R;
    else-----
        PWM_L<=RMS_L;
        PWM_R<=RMS_R;
    end if;
elseif(edot_region="11") then
    if(Edot_sign='0') then-
-----PMB EDOT-----
        PWM_L<=LS_L;
        PWM_R<=LS_R;
    else-----
        PWM_L<=RMS_L;
        PWM_R<=RMS_R;
    end if;
end if;
end if;
end if;

elseif(error_region="10") then
-----PMS AND PMB FOR ERROR-----
    if(error_sign='0') then
        if(error_mem1>="0111") then-----
--PMS ERROR-----
            if(edot_region="00") then---
---ZERO EDOT-----
                PWM_L<=LMS_L;
                PWM_R<=LMS_R;
            elseif(edot_region="01") then
                if(Edot_sign='0') then-
-----PS EDOT-----
                    PWM_L<=LMS_L;
                    PWM_R<=LMS_R;
                else-----
-----NS EDOT-----
                    PWM_L<=ZO_L;

```

```

PWM_R<=ZO_R;
end if;
elsif(edot_region="10") then
    if(Edot_sign='0') then-
-----PMS EDOT-----
PWM_L<=LMS_L;
PWM_R<=LMS_R;
else-----
-----NMS EDOT-----
PWM_L<=RS_L;
PWM_R<=RS_R;
end if;
elsif(edot_region="11") then
    if(Edot_sign='0') then-
-----PMB EDOT-----
PWM_L<=LMS_L;
PWM_R<=LMS_R;
else-----
-----NMB EDOT-----
PWM_L<=RS_L;
PWM_R<=RS_R;
end if;
end if;
else-----
if(edot_region="00") then---
PWM_L<=LMB_L;
PWM_R<=LMB_R;
elsif(edot_region="01") then
    if(Edot_sign='0') then-
-----PS EDOT-----
PWM_L<=LMB_L;
PWM_R<=LMB_R;
else-----
-----NS EDOT-----
PWM_L<=LMS_L;
PWM_R<=LMS_R;
end if;
elsif(edot_region="10") then
    if(Edot_sign='0') then-
-----PMS EDOT-----
PWM_L<=LMB_L;
PWM_R<=LMB_R;
else-----
-----NMS EDOT-----
PWM_L<=LS_L;
PWM_R<=LS_R;
end if;
elsif(edot_region="11") then
    if(Edot_sign='0') then-
-----PMB EDOT-----
PWM_L<=LMB_L;
PWM_R<=LMB_R;
else-----
-----NMB EDOT-----
PWM_L<=ZO_L;

```

```

                                PWM_R<=ZO_R;
                                end if;
                                end if;
                                end if;
-----NMS AND NMB FOR ERROR-----
-----
                                else
                                if(error_mem1>="0111") then-----
-NMS ERROR-----
                                if(edot_region="00") then----
---ZERO EDOT-----
                                PWM_L<=RMS_L;
                                PWM_R<=RMS_R;
                                elsif(edot_region="01") then
                                if(Edot_sign='0') then-
-----PS EDOT-----
                                PWM_L<=ZO_L;
                                PWM_R<=ZO_R;
                                else-----
-----NS EDOT-----
                                PWM_L<=RMS_L;
                                PWM_R<=RMS_R;
                                end if;
                                elsif(edot_region="10") then
                                if(Edot_sign='0') then-
-----PMS EDOT-----
                                PWM_L<=LS_L;
                                PWM_R<=LS_R;
                                else-----
-----NMS EDOT-----
                                PWM_L<=RMS_L;
                                PWM_R<=RMS_R;
                                end if;
                                elsif(edot_region="11") then
                                if(Edot_sign='0') then-
-----PMB EDOT-----
                                PWM_L<=LS_L;
                                PWM_R<=LS_R;
                                else-----
-----NMB EDOT-----
                                PWM_L<=RMS_L;
                                PWM_R<=RMS_R;
                                end if;
                                end if;
                                else-----
-NMB ERROR-----
                                if(edot_region="00") then----
---ZERO EDOT-----
                                PWM_L<=RMB_L;
                                PWM_R<=RMB_R;
                                elsif(edot_region="01") then
                                if(Edot_sign='0') then-
-----PS EDOT-----
                                PWM_L<=RMS_L;
                                PWM_R<=ZO_R;
                                else-----
-----NS EDOT-----

```

```

                                PWM_L<=RMB_L;
                                PWM_R<=RMB_R;
                                end if;
elseif(edot_region="10") then
                                if(Edot_sign='0') then-
-----PMS EDOT-----
                                PWM_L<=RS_L;
                                PWM_R<=RS_R;
                                else-----
-----NMS EDOT-----
                                PWM_L<=RMB_L;
                                PWM_R<=RMB_R;
                                end if;
elseif(edot_region="11") then
                                if(Edot_sign='0') then-
-----PMB EDOT-----
                                PWM_L<=ZO_L;
                                PWM_R<=ZO_R;
                                else-----
-----NMB EDOT-----
                                PWM_L<=RMB_L;
                                PWM_R<=RMB_R;
                                end if;
                                end if;
                                end if;
                                end if;
                                end if;

else
-----PB FOR ERROR-----
-----
                                if(error_sign='0') then-----
PB ERROR-----
                                if(edot_region="00") then----
-----ZERO EDOT-----
                                PWM_L<=LMB_L;
                                PWM_R<=LMB_R;
                                elseif(edot_region="01") then
                                if(Edot_sign='0') then-
-----PS EDOT-----
                                PWM_L<=LMB_L;
                                PWM_R<=LMB_R;
                                else-----
-----NS EDOT-----
                                PWM_L<=LMS_L;
                                PWM_R<=LMS_R;
                                end if;
                                elseif(edot_region="10") then
                                if(Edot_sign='0') then-
-----PMS EDOT-----
                                PWM_L<=LMB_L;
                                PWM_R<=LMB_R;

```

```

-----NMS EDOT-----
else-----
        PWM_L<=LS_L;
        PWM_R<=LS_R;
    end if;
elseif(edot_region="11") then
    if(Edot_sign='0') then-
-----PMB EDOT-----
        PWM_L<=LMB_L;
        PWM_R<=LMB_R;
    else-----
-----NMB EDOT-----
        PWM_L<=ZO_L;
        PWM_R<=ZO_R;
    end if;
    end if;
-----NB FOR ERROR-----
else-----
NB ERROR-----
    if(edot_region="00") then---
        PWM_L<=RMB_L;
        PWM_R<=RMB_R;
    elseif(edot_region="01") then
        if(Edot_sign='0') then-
-----PS EDOT-----
            PWM_L<=RMS_L;
            PWM_R<=RMS_R;
        else-----
-----NS EDOT-----
            PWM_L<=RMB_L;
            PWM_R<=RMB_R;
        end if;
    elseif(edot_region="10") then
        if(Edot_sign='0') then-
-----PMS EDOT-----
            PWM_L<=RS_L;
            PWM_R<=RS_R;
        else-----
-----NMS EDOT-----
            PWM_L<=RMB_L;
            PWM_R<=RMB_R;
        end if;
    elseif(edot_region="11") then
        if(Edot_sign='0') then-
-----PMB EDOT-----
            PWM_L<=ZO_L;
            PWM_R<=ZO_R;
        else-----
-----NMB EDOT-----
            PWM_L<=RMB_L;
            PWM_R<=RMB_R;
        end if;
    end if;
end if;
end if;

```

```

--          end if;
    else
        Last_Error:=Error_i;
        Last_Error_sign:=Error_sign;
--        if(flag_reset="111") then
--            flag_reset:="110";
--        end if;
--        if(flag_reset="101") then
--            flag_reset:="100";
--        end if;
--        if(flag_reset="011") then
--            flag_reset:="010";
--        end if;
--        if(flag_reset="001") then
--            flag_reset:="000";
--        end if;
        end if;
    end if;
end process;

end Behavioral;

```


7. LCD Interface Module

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Sensor_Display4 is
    Port(
        CLK: in STD_LOGIC;
        Trigger: in STD_LOGIC;
        Data_L: in STD_LOGIC_VECTOR(21 downto 0);
        Data_R: in STD_LOGIC_VECTOR(21 downto 0);
        Data_L1: in STD_LOGIC_VECTOR(21 downto 0);
        Data_R1: in STD_LOGIC_VECTOR(21 downto 0);
        Data0: out STD_LOGIC;
        Data1: out STD_LOGIC;
        Data2: out STD_LOGIC;
        Data3: out STD_LOGIC;
        Data4: out STD_LOGIC;
        Data5: out STD_LOGIC;
        Data6: out STD_LOGIC;
        Data7: out STD_LOGIC;
        EN: out STD_LOGIC;
        RS: out STD_LOGIC);
end Sensor_Display4;

architecture Behavioral of Sensor_Display4 is
    signal count:STD_LOGIC_VECTOR(13 downto 0);
    signal count1:STD_LOGIC_VECTOR(3 downto 0);
    type STATE_TYPE is (State_Clear_0, State_Clear, State_Clear_1,
        State0_0, State0, State1, State2_0, State2, State3, State4_0, State4,
        State5, State6_0, State6, State7, State8_0, State8, State9,
        State10_0, State10, State11, State12_0, State12, State13,
        State14_0, State14, State15, State16_0, State16, State17, State18_0,
        State18, State19,
        State20_0, State20, State21, State22_0, State22, State23,
        State24_0, State24, State25, State26_0, State26, State27, State28_0,
        State28, State29,
        State30_0, State30, State31, State32_0, State32, State33,
        State34_0, State34, State35, State36_0, State36, State37, State38_0,
        State38, State39,
        State40_0, State40, State41, State42_0, State42, State43,
        State44_0, State44, State45, State46_0, State46, State47, State48_0,
        State48, State49,
        State50_0, State50, State51, State52_0, State52, State53,
        State54_0, State54, State55, State56_0, State56, State57, State58_0,
        State58, State59,
```

```

        State60_0, State60, State61, State62_0, State62, State63,
        State64_0, State64, State65, State66_0, State66, State67);
        signal CS, NS: STATE_TYPE;

```

```

begin
    process(CLK)
    begin
        if(CLK='1' and CLK'EVENT) then
            count<=count+1;
            if(count="11111111111111") then
                if(CS=State_Clear) then
                    count1<=count1+1;
                    if(count1="1111") then
                        CS<=NS;
                        count1<="0000";
                    end if;
                else
                    CS<=NS;
                end if;
                count<="0000000000000000";
            end if;
        end if;
    end process;

```

```

COMB_PROC: process (CS, Trigger)
    variable dis:STD_LOGIC_VECTOR(3 downto 0);
begin
    case CS is
        when State_Clear_0 =>
            Data7<='0';
            Data6<='0';
            Data5<='0';
            Data4<='0';
            Data3<='0';
            Data2<='0';
            Data1<='0';
            Data0<='1';
            EN<='1';
            RS<='0';
            if(Trigger='1') then
                NS<=State_Clear;
            else
                NS<=State_Clear_0;
            end if;

            when State_Clear =>
                Data7<='0';
                Data6<='0';
                Data5<='0';
                Data4<='0';
                Data3<='0';
                Data2<='0';
                Data1<='0';
                Data0<='1';

```

```

    EN<='0';
    RS<='0';
    NS<=State_Clear_1;

when State_Clear_1 =>
    Data7<='0';
    Data6<='0';
    Data5<='0';
    Data4<='0';
    Data3<='0';
    Data2<='0';
    Data1<='0';
    Data0<='1';
    EN<='1';
    RS<='0';
    NS<=State0_0;

when State0_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='0';
    NS<=State0;

when State0 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='0';
    RS<='0';
    NS<=State1;

when State1 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='1';
    Data2<='0';
    Data1<='0';

```

```

        Data0<='0';
        EN<='1';
        RS<='0';
        NS<=State2_0;

when State2_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='1';
    Data1<='1';
    Data0<='0';
    EN<='1';
    RS<='0';
    NS<=State2;

when State2 =>
    Data7<='0';
    Data6<='0';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='1';
    Data1<='1';
    Data0<='0';
    EN<='0';
    RS<='0';
    NS<=State3;

when State3 =>
    Data7<='0';
    Data6<='0';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='1';
    Data1<='1';
    Data0<='0';
    EN<='1';
    RS<='0';
    NS<=State4_0;

when State4_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='0';
    Data4<='0';
    Data3<='0';
    Data2<='1';
    Data1<='1';

```

```

        Data0<='0';
        EN<='1';
        RS<='0';
        NS<=State4;

when State4 =>
    Data7<='0';
    Data6<='0';
    Data5<='0';
    Data4<='0';
    Data3<='0';
    Data2<='1';
    Data1<='1';
    Data0<='0';
    EN<='0';
    RS<='0';
    NS<=State5;

when State5 =>
    Data7<='0';
    Data6<='0';
    Data5<='0';
    Data4<='0';
    Data3<='0';
    Data2<='1';
    Data1<='1';
    Data0<='0';
    EN<='1';
    RS<='0';
    NS<=State6_0;

when State6_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_L(21);
    Data0<=Data_L(20);
    EN<='1';
    RS<='1';
    NS<=State6;

when State6 =>
    RS<='1';
    Data7<='0';
    Data6<='0';
    Data5<='1';

```

```

Data4<='1';
Data3<='0';
Data2<='0';
Data1<=Data_L(21);
Data0<=Data_L(20);
EN<='0';
NS<=State7;

when State7 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_L(21);
    Data0<=Data_L(20);
    EN<='1';
    RS<='1';
    NS<=State8_0;

when State8_0 =>
    dis(3 downto 0):=Data_L(19 downto 16);
    if(Data_L(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State8;

when State8 =>
    RS<='1';
    dis(3 downto 0):=Data_L(19 downto 16);
    if(Data_L(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;

```

```

else
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
end if;
Data3<=dis(3);
Data2<=dis(2);
Data1<=dis(1);
Data0<=dis(0);
EN<='0';
NS<=State9;

when State9 =>
    dis(3 downto 0):=Data_L(19 downto 16);
    if(Data_L(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State10_0;

when State10_0 =>
    dis(3 downto 0):=Data_L(15 downto 12);
    if(Data_L(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);

```

```

EN<='1';
RS<='1';
NS<=State10;

when State10 =>
    RS<='1';
    dis(3 downto 0):=Data_L(15 downto 12);
    if(Data_L(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State11;

when State11 =>
    dis(3 downto 0):=Data_L(15 downto 12);
    if(Data_L(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State12_0;

when State12_0 =>
    dis(3 downto 0):=Data_L(11 downto 8);
    if(Data_L(11 downto 8)>"1001") then
        Data7<='0';

```



```

        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State12;

when State12 =>
    RS<='1';
    dis(3 downto 0):=Data_L(11 downto 8);
    if(Data_L(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State13;

when State13 =>
    dis(3 downto 0):=Data_L(11 downto 8);
    if(Data_L(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);

```

```

Data2<=dis(2);
Data1<=dis(1);
Data0<=dis(0);
EN<='1';
RS<='1';
NS<=State14_0;

when State14_0 =>
    dis(3 downto 0):=Data_L(7 downto 4);
    if(Data_L(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State14;

when State14 =>
    RS<='1';
    dis(3 downto 0):=Data_L(7 downto 4);
    if(Data_L(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State15;

when State15 =>

```

```

dis(3 downto 0):=Data_L(7 downto 4);
if(Data_L(7 downto 4)>"1001") then
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    dis:=dis-9;
else
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
end if;
Data3<=dis(3);
Data2<=dis(2);
Data1<=dis(1);
Data0<=dis(0);
EN<='1';
RS<='1';
NS<=State16_0;

when State16_0 =>
    dis(3 downto 0):=Data_L(3 downto 0);
    if(Data_L(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State16;

when State16 =>
    RS<='1';
    dis(3 downto 0):=Data_L(3 downto 0);
    if(Data_L(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else

```

```

        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State17;

when State17 =>
    dis(3 downto 0):=Data_L(3 downto 0);
    if(Data_L(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State18_0;

when State18_0 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State18;

when State18 =>
    RS<='1';
    Data7<='0';
    Data6<='1';
    Data5<='0';

```

```

        Data4<='0';
        Data3<='1';
        Data2<='0';
        Data1<='0';
        Data0<='0';
        EN<='0';
        NS<=State19;

when State19 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State20_0;

when State20_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='0';
    Data3<='1';
    Data2<='1';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State20;

when State20=>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='0';
    Data3<='1';
    Data2<='1';
    Data1<='0';
    Data0<='0';
    EN<='0';
    RS<='1';
    NS<=State21;

when State21 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';

```

```

        Data4<='0';
        Data3<='1';
        Data2<='1';
        Data1<='0';
        Data0<='0';
        EN<='1';
        RS<='1';
        NS<=State22_0;

when State22_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_R(21);
    Data0<=Data_R(20);
    EN<='1';
    RS<='1';
    NS<=State22;

when State22 =>
    RS<='1';
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_R(21);
    Data0<=Data_R(20);
    EN<='0';
    NS<=State23;

when State23 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_R(21);
    Data0<=Data_R(20);
    EN<='1';
    RS<='1';
    NS<=State24_0;

when State24_0 =>
    dis(3 downto 0):=Data_R(19 downto 16);

```

```

    if(Data_R(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State24;

when State24 =>
    RS<='1';
    dis(3 downto 0):=Data_R(19 downto 16);
    if(Data_R(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State25;

when State25 =>
    dis(3 downto 0):=Data_R(19 downto 16);
    if(Data_R(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';

```

```

end if;
Data3<=dis(3);
Data2<=dis(2);
Data1<=dis(1);
Data0<=dis(0);
EN<='1';
RS<='1';
NS<=State26_0;

when State26_0 =>
    dis(3 downto 0):=Data_R(15 downto 12);
    if(Data_R(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State26;

when State26 =>
    RS<='1';
    dis(3 downto 0):=Data_R(15 downto 12);
    if(Data_R(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State27;

```



```

when State27 =>
    dis(3 downto 0):=Data_R(15 downto 12);
    if(Data_R(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State28_0;

```

```

when State28_0 =>
    dis(3 downto 0):=Data_R(11 downto 8);
    if(Data_R(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State28;

```

```

when State28 =>
    RS<='1';
    dis(3 downto 0):=Data_R(11 downto 8);
    if(Data_R(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
    end if;

```

```

        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State29;

when State29 =>
    dis(3 downto 0):=Data_R(11 downto 8);
    if(Data_R(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State30_0;

when State30_0 =>
    dis(3 downto 0):=Data_R(7 downto 4);
    if(Data_R(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);

```

```

Data0<=dis(0);
EN<='1';
RS<='1';
NS<=State30;

when State30 =>
    RS<='1';
    dis(3 downto 0):=Data_R(7 downto 4);
    if(Data_R(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State31;

when State31 =>
    dis(3 downto 0):=Data_R(7 downto 4);
    if(Data_R(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State32_0;

when State32_0 =>
    dis(3 downto 0):=Data_R(3 downto 0);
    if(Data_R(3 downto 0)>"1001") then

```

```

        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State32;

when State32 =>
    RS<='1';
    dis(3 downto 0):=Data_R(3 downto 0);
    if(Data_R(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State33;

when State33 =>
    dis(3 downto 0):=Data_R(3 downto 0);
    if(Data_R(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;

```

```

        Data3<=dis(3);
        Data2<=dis(2);
        Data1<=dis(1);
        Data0<=dis(0);
        EN<='1';
        RS<='1';
        NS<=State34_0;

when State34_0 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State34;

when State34 =>
    RS<='1';
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='0';
    NS<=State35;

when State35 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State36_0;

when State36_0 =>
    Data7<='1';

```

```

        Data6<=' 1' ;
        Data5<=' 0' ;
        Data4<=' 0' ;
        Data3<=' 0' ;
        Data2<=' 0' ;
        Data1<=' 0' ;
        Data0<=' 0' ;
        EN<=' 1' ;
        RS<=' 0' ;
        NS<=State36;

when State36=>
    Data7<=' 1' ;
    Data6<=' 1' ;
    Data5<=' 0' ;
    Data4<=' 0' ;
    Data3<=' 0' ;
    Data2<=' 0' ;
    Data1<=' 0' ;
    Data0<=' 0' ;
    EN<=' 0' ;
    RS<=' 0' ;
    NS<=State37;

when State37 =>
    Data7<=' 1' ;
    Data6<=' 1' ;
    Data5<=' 0' ;
    Data4<=' 0' ;
    Data3<=' 0' ;
    Data2<=' 0' ;
    Data1<=' 0' ;
    Data0<=' 0' ;
    EN<=' 1' ;
    RS<=' 0' ;
    NS<=State38_0;

when State38_0 =>
    Data7<=' 0' ;
    Data6<=' 0' ;
    Data5<=' 1' ;
    Data4<=' 1' ;
    Data3<=' 0' ;
    Data2<=' 0' ;
    Data1<=Data_L1(21);
    Data0<=Data_L1(20);
    EN<=' 1' ;
    RS<=' 1' ;
    NS<=State38;

when State38 =>
    RS<=' 1' ;

```

```

Data7<='0';
Data6<='0';
Data5<='1';
Data4<='1';
Data3<='0';
Data2<='0';
Data1<=Data_L1(21);
Data0<=Data_L1(20);
EN<='0';
NS<=State39;

when State39 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_L1(21);
    Data0<=Data_L1(20);
    EN<='1';
    RS<='1';
    NS<=State40_0;

when State40_0 =>
    dis(3 downto 0):=Data_L1(19 downto 16);
    if(Data_L1(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State40;

when State40 =>
    RS<='1';
    dis(3 downto 0):=Data_L1(19 downto 16);
    if(Data_L1(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';

```

```

        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State41;

when State41 =>
    dis(3 downto 0):=Data_L1(19 downto 16);
    if(Data_L1(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State42_0;

when State42_0 =>
    dis(3 downto 0):=Data_L1(15 downto 12);
    if(Data_L1(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);

```



```

    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State42;

when State42 =>
    RS<='1';
    dis(3 downto 0):=Data_L1(15 downto 12);
    if(Data_L1(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State43;

when State43 =>
    dis(3 downto 0):=Data_L1(15 downto 12);
    if(Data_L1(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State44_0;

when State44_0 =>

```

```

dis(3 downto 0):=Data_L1(11 downto 8);
if(Data_L1(11 downto 8)>"1001") then
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    dis:=dis-9;
else
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
end if;
Data3<=dis(3);
Data2<=dis(2);
Data1<=dis(1);
Data0<=dis(0);
EN<='1';
RS<='1';
NS<=State44;

when State44 =>
    RS<='1';
    dis(3 downto 0):=Data_L1(11 downto 8);
    if(Data_L1(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State45;

when State45 =>
    dis(3 downto 0):=Data_L1(11 downto 8);
    if(Data_L1(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';

```

```

        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State46_0;

when State46_0 =>
    dis(3 downto 0):=Data_L1(7 downto 4);
    if(Data_L1(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State46;

when State46 =>
    RS<='1';
    dis(3 downto 0):=Data_L1(7 downto 4);
    if(Data_L1(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State47;

```

```

when State47 =>
    dis(3 downto 0):=Data_L1(7 downto 4);
    if(Data_L1(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State48_0;

```

```

when State48_0 =>
    dis(3 downto 0):=Data_L1(3 downto 0);
    if(Data_L1(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State48;

```

```

when State48 =>
    RS<='1';
    dis(3 downto 0):=Data_L1(3 downto 0);
    if(Data_L1(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';

```

```

        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State49;

when State49 =>
    dis(3 downto 0):=Data_L1(3 downto 0);
    if(Data_L1(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State50_0;

when State50_0 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State50;

when State50 =>
    RS<='1';

```

```

        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        Data3<='1';
        Data2<='0';
        Data1<='0';
        Data0<='0';
        EN<='0';
        NS<=State51;

when State51 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State52_0;

when State52_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='0';
    Data3<='1';
    Data2<='1';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State52;

when State52=>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='0';
    Data3<='1';
    Data2<='1';
    Data1<='0';
    Data0<='0';
    EN<='0';
    RS<='1';
    NS<=State53;

when State53 =>

```

```

        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='0';
        Data3<='1';
        Data2<='1';
        Data1<='0';
        Data0<='0';
        EN<='1';
        RS<='1';
        NS<=State54_0;

when State54_0 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_R1(21);
    Data0<=Data_R1(20);
    EN<='1';
    RS<='1';
    NS<=State54;

when State54 =>
    RS<='1';
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_R1(21);
    Data0<=Data_R1(20);
    EN<='0';
    NS<=State55;

when State55 =>
    Data7<='0';
    Data6<='0';
    Data5<='1';
    Data4<='1';
    Data3<='0';
    Data2<='0';
    Data1<=Data_R1(21);
    Data0<=Data_R1(20);
    EN<='1';
    RS<='1';
    NS<=State56_0;

```

```

when State56_0 =>
    dis(3 downto 0):=Data_R1(19 downto 16);
    if(Data_R1(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State56;

when State56 =>
    RS<='1';
    dis(3 downto 0):=Data_R1(19 downto 16);
    if(Data_R1(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State57;

when State57 =>
    dis(3 downto 0):=Data_R1(19 downto 16);
    if(Data_R1(19 downto 16)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';

```



```

        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State58_0;

when State58_0 =>
    dis(3 downto 0):=Data_R1(15 downto 12);
    if(Data_R1(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State58;

when State58 =>
    RS<='1';
    dis(3 downto 0):=Data_R1(15 downto 12);
    if(Data_R1(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);

```

```

EN<='0';
NS<=State59;

when State59 =>
    dis(3 downto 0):=Data_R1(15 downto 12);
    if(Data_R1(15 downto 12)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State60_0;

when State60_0 =>
    dis(3 downto 0):=Data_R1(11 downto 8);
    if(Data_R1(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State60;

when State60 =>
    RS<='1';
    dis(3 downto 0):=Data_R1(11 downto 8);
    if(Data_R1(11 downto 8)>"1001") then
        Data7<='0';

```

```

        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State61;

when State61 =>
    dis(3 downto 0):=Data_R1(11 downto 8);
    if(Data_R1(11 downto 8)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State62_0;

when State62_0 =>
    dis(3 downto 0):=Data_R1(7 downto 4);
    if(Data_R1(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;

```

```

    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State62;

when State62 =>
    RS<='1';
    dis(3 downto 0):=Data_R1(7 downto 4);
    if(Data_R1(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State63;

when State63 =>
    dis(3 downto 0):=Data_R1(7 downto 4);
    if(Data_R1(7 downto 4)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State64_0;

```

```

when State64_0 =>
    dis(3 downto 0):=Data_R1(3 downto 0);
    if(Data_R1(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State64;

```

```

when State64 =>
    RS<='1';
    dis(3 downto 0):=Data_R1(3 downto 0);
    if(Data_R1(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';
        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='0';
    NS<=State65;

```

```

when State65 =>
    dis(3 downto 0):=Data_R1(3 downto 0);
    if(Data_R1(3 downto 0)>"1001") then
        Data7<='0';
        Data6<='1';
        Data5<='0';
        Data4<='0';
        dis:=dis-9;
    else
        Data7<='0';
        Data6<='0';

```

```

        Data5<='1';
        Data4<='1';
    end if;
    Data3<=dis(3);
    Data2<=dis(2);
    Data1<=dis(1);
    Data0<=dis(0);
    EN<='1';
    RS<='1';
    NS<=State66_0;

when State66_0 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    NS<=State66;

when State66 =>
    RS<='1';
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='0';
    NS<=State67;

when State67 =>
    Data7<='0';
    Data6<='1';
    Data5<='0';
    Data4<='0';
    Data3<='1';
    Data2<='0';
    Data1<='0';
    Data0<='0';
    EN<='1';
    RS<='1';
    if (Trigger='1') then
        NS<=State35;
    else
        NS<=State_Clear_0;
    end if;
end when;

```

```
end if;  
  
end case;  
end process;  
  
end Behavioral;
```

BIBLIOGRAPHY

[1] Burke Daniel M. "A Layered Fuzzy Motion Controller for a Mobile Robot". MS Thesis, Wright State University 1992.

[2] Brooks Rodney A. "A robust layered control system for a mobile robot". IEEE Journal of Robotics and Automation, RA-2: 14-23, 1986.

[3] Anthony Del Sorbo. "The Development of a Modular Multi-Processing Autonomous Mobile Platform". MS Thesis, Wright State University, 2004.

[4] Friendly Robotics. RoboMower [Online] Available
<http://www.friendlyrobotics.com/NewSite/robomow.htm>

[5] Examples of Fuzzy Logic. [Online] Available
<http://www.atp.rub.de/DynLAB/dynlabmodules/Examples/FuzzyLogic/FuzzyAnd.html>

[6] Devantech Ltd. SRF04-Ultrasonic Ranger. Technical Specification. [Online]
Available http://www.robot-electronics.co.uk/htm/srf04_timing.shtml

[7] Active Robotics. PeopleBot® [Online] Available
<http://www.activrobots.com/ROBOTS/peoplebot.html>

- [8] SHARP Infrared sensor, GP2D15, Datasheet. [Online] Available <http://www.informatik.uni-oldenburg.de/~trebla/GP2D12/gp2d12.pdf> 87
- [9] Introduction to Fuzzy Control. [Online] Available http://www.cs.wright.edu/people/faculty/krattan/courses/419/outline_fall05.htm
- [10] Introduction to automation. [Online] Available: <http://science.howstuffworks.com>
- [11] Sameep Singh. “Implementation of a Fuzzy Logic Controller on a FPGA using VHDL”. MS Thesis, Wright State University, 2002.
- [12] Priti Gaonkar. “Implementation of a Layered Fuzzy Controller for an autonomous mobile robot”. MS Thesis, Wright State University, 2005.
- [13] Xilinx Virtex-II Pro FPGA by Avnet
- [14] Robotics and automation. [Online] Available: <http://roboticsconnection.com>