# FPGA Implemented Fuzzy Logic Based Intelligent Load Management System

Thesis submitted in partial fulfillment of the requirement for

the award of degree of

**Master of Technology**

in

**VLSI Design & CAD**

Submitted by

**Narender Kumar**

**(600861031)**

Under the supervision of

**Dr. Hardeep Singh**

**Assistant Professor**

**Thapar University**

Department of Electronics & Communication Engineering

Thapar University

Patiala 147004(Punjab). India

**July2010**

# CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "**FPGA Implemented Fuzzy Logic Based Intelligent Load Management System**", in partial fulfillment of the requirements for the award of degree **of Master of Technology in VLSI Design & CAD** submitted **in Electronics and Communication Engineering Department** of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Hardeep Singh** and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

**Date:** 13.7.10.

**Narender Kumar**
(Signature of the student)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

**Dr. Hardeep Singh**
**(Asst. Professor)**
**Department of ECE**
**Thapar University, Patiala**

**Dr. A.K Chatterjee** 13.7.10.
**(Prof. & Head ECED)**
**Thapar University, Patiala**

**Dr. R.K. Sharma** 13.7.10
**(Dean of Academic Affairs)**
**Thapar University, Patiala**

# ACKNOWLEDGEMENT

To achieve success in any work, guidance plays an important role. It makes us put the right amount of energy in the right direction and at right time to obtain the desired result. I express my sincere gratitude to my supervisor , **Dr. Hardeep Singh**, Assistant Professor Department of Electronics and Communication Engineering, for giving valuable guidance during the course of this investigation, for his ever encouraging and timely moral support. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. His enormous knowledge and intelligence always helped me unconditionally to solve various problems.

I express my feelings of thanks to the entire faculty and staff of Electronics and Communication Engineering Department, Thapar University, Patiala for their help, inspiration and moral support, which went a long way in successfully completion of my Thesis.

I am deeply indebted to my parents, brother Raman and my wife Geeta for their inspiration and ever encouraging moral support, which enabled me to pursue my studies. I am also thankful to the authors whose works I have consulted and quoted in this work. Last but not the least I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. Lastly I would like to thank my cousin Lalit Upadhayay for always supporting me and being there on my side whenever I needed.

**Narender Kumar**
**(Reg. No. 600861031)**

# ABSTRACT

Power management is a very important aspect of any network. With power management, the capacity, the throughput and battery lifetime could be increased, and latency could be reduced. But all of these advantages may be hindered depending on the power management employed and depending on the network configuration. The issues and advantages involved in the design of power control and management schemes along with a novel approach to power management by utilizing modern control theory to achieve the low-level power control. Captive power refers to electric power generation from a unit set up by an organization for its exclusive consumption. A number of organizations are now increasingly relying on their own generation rather than on grid supply primarily for the reasons of inadequate grid supply as well as its poor quality and reliability.

The use of captive power is on the rise across the globe and most of the practical power plants are run sub-optimally by operators. This calls for low-cost dedicated automatic controllers for the efficient management of captive power. Traditionally, the operation and management of Diesel Generator (DG) sets in captive power plants is done manually. Experienced though, the plant operators are prone to all the pitfalls of manual control, and typically their inadequate decisions may not only lead to inefficient use of available captive power but also create dissatisfaction among its users. To obviate such shortcomings, it is natural to conceive concept of Fuzzy Logic System for such applications, which is capable of taking adequate decisions for the efficient management of captive power, as per the rule base suggested by expert opinions of the experienced D. G. Set operators.

# CONTENTS

# LIST OF FIGURES

# Chapter 1: Introduction

## 1. Introduction to the Problem

The purpose of this project is to intelligently control the distribution of Power supply from Diesel Generator (DG) set in a residential college or university (i.e. Thapar University). Some work has been done of various schemes for load shedding interfacing with SCADA [1] and transmission of power management in ad hoc networks [2]. Though SCADA systems are meant to be information gatherers as well as for exercising control. For this an FPGA is suggested that will automate the process using Fuzzy Logic [3, 4]. The system will be time and load dependent and will be able to supply power to the required area of the campus as output. Currently, human effort is required to decide the areas of the campus as output. During power cuts, the campus is left with a D.G. set to back it up for power; otherwise there is no requirement of power cut in any specific area. This system is supposed to work only during mains power cut. Decision made by human being may be biased. Moreover extra labor is required to make the decision for the distribution of the Power Supply. Thus an intelligent, automated and unbiased system is required.

An ad-hoc manual system of load shedding is plagued by the fact that the load quantum and time of load (e.g. morning, evening, night, late night, etc.) can at best be fuzzily specified by the operator. Such linguistic imprecision (rather than the presence of random variables) can be easily handled by fuzzy logic [5]. In contrast to a manual load shedding arrangement, the automated system is capable of making intelligent decisions to make optimum use of power as also to result in maximum user satisfaction in case of over load on the D.G. set. The theoretical analysis of this system revealed that while dealing with load shedding, the operator is able to specify the 'Time' and 'Load' at best only fuzzily.

TIME & LOAD, being fuzzily specified, become the two input decision making variables for the FIS (Fuzzy Inference System). These are fuzzified into three and five fuzzy sets respectively. The TIME variable is fuzzified into three fuzzy sets: MNight (Morning Night), Day (Day) & ENight (Evening Night). Similarly, LOAD input variable is fuzzified into five appropriate fuzzy sets: Low (Low), Medium (Medium), High (High), VHigh (Very High) & XHigh (Extremely High). Of course, when LOAD will be Low no area should face power cut. Due to larger spans between fuzzy sets of TIME, these are taken to be of trapezoidal shapes while fuzzy sets for LOAD are taken to be of triangular

shapes for reasons of economic representation. The output variable is taken as the command to relays controlling different sections. As the system will designed for implementation on an FPGA chip [6,7] so the rule matrix is designed for the minimum possible rules. To that end, those rules which give similar output are aggregated using OR operation. VHDL [8]is used for programming & implementation of this fuzzy system on Xilinx FPGA chip.

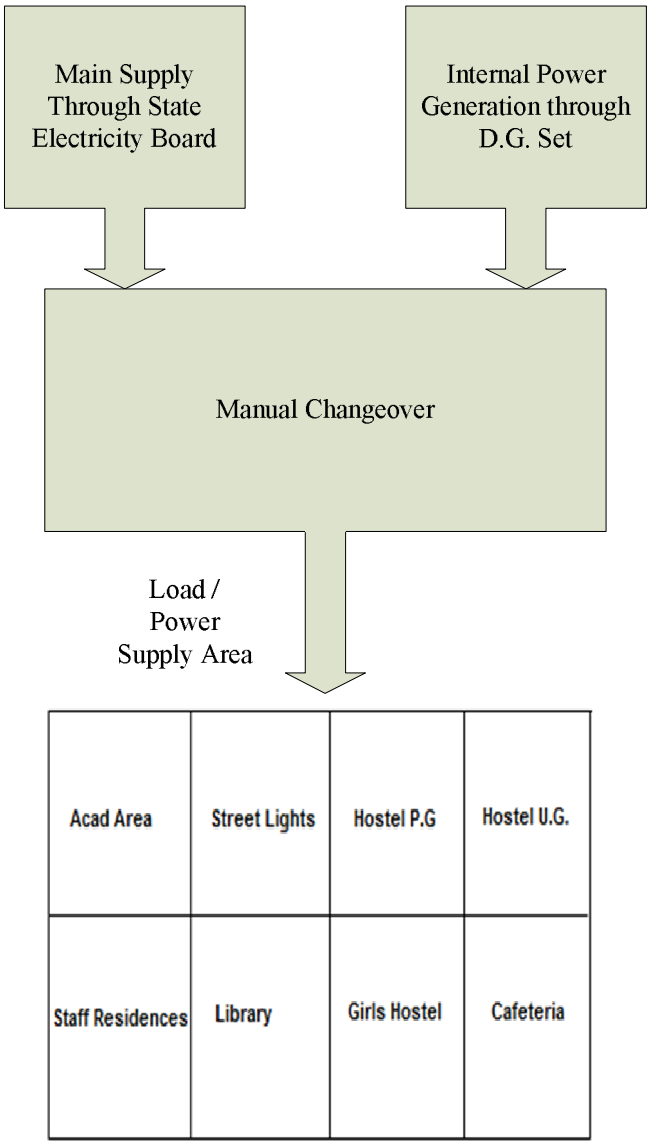


**Fig.1: Block Diagram of the Existing System**

**Proposed System**

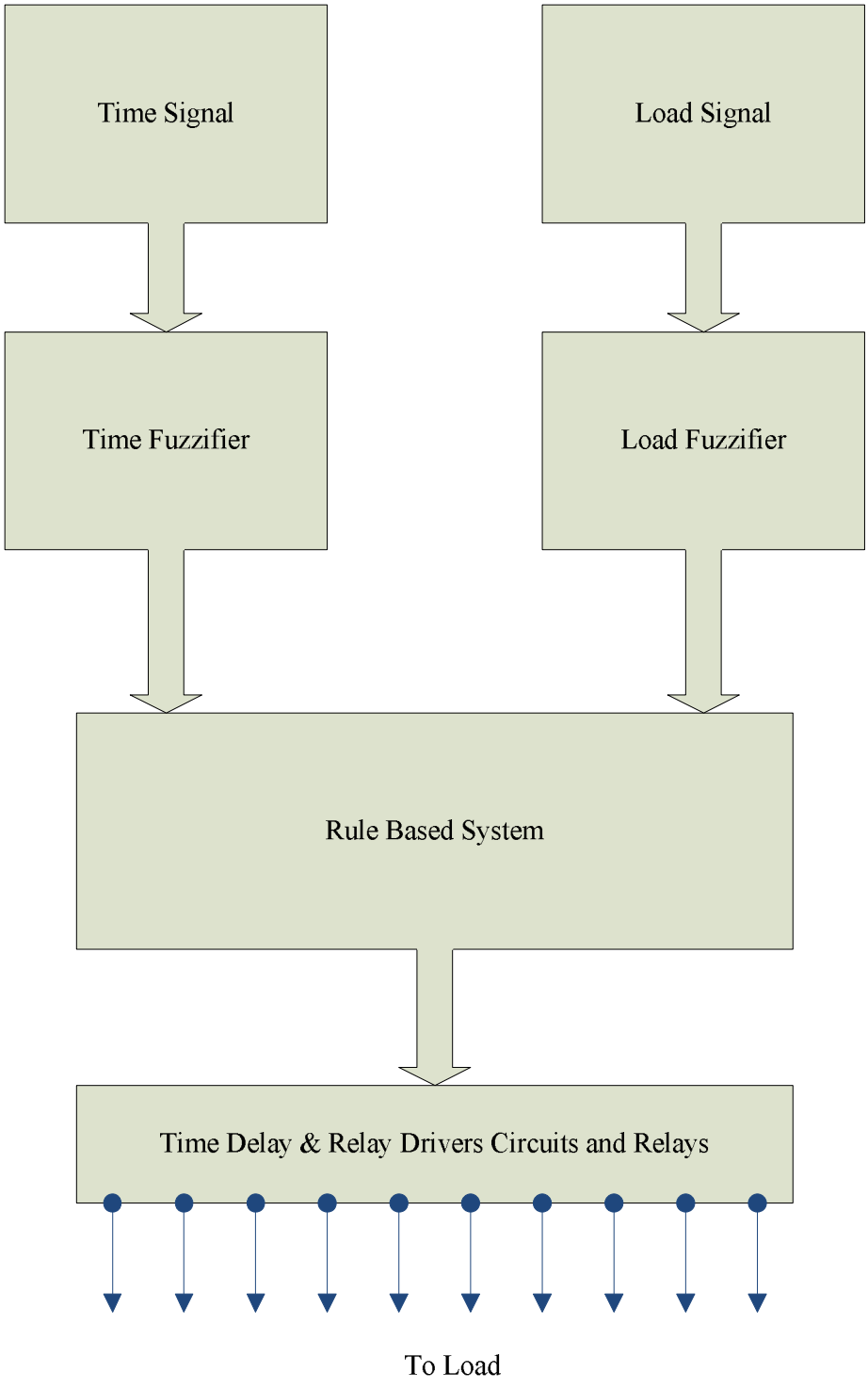


**Fig.2: Block Diagram of the Proposed System**

# Chapter 3: Fuzzy Logic

## 2.1 Introduction

The concept of Fuzzy Logic (FL) was conceived at the beginning of the 70s by Lotfi Zadeh, a professor at the University of California at Berkley [4] intent not as a control methodology, but as a way of processing data by allowing partial set membership rather than crisp set membership or non-membership. This approach to set theory was not applied to control systems until the 70's due to insufficient small-computer capability prior to that time. Professor Zadeh reasoned that people do not require precise, numerical information input, and yet they are capable of highly adaptive control.[9,10] If feedback controllers could be programmed to accept noisy, imprecise input, they would be much more effective and perhaps easier to implement. In this context, FL is a problem-solving control system methodology that lends itself to implementation in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems.[11,12] It can be implemented in hardware, software, or a combination of both. FL provides a simple way to arrive at a definite conclusion based upon vague, ambiguous, imprecise, noisy, or missing input information. FL's approach to control problems mimics how a person would make decisions, only much faster. .[13,14]

In recent years, the number and variety of applications of fuzzy logic have increased significantly. The applications range from consumer products such as cameras, camcorders, washing machines, and microwave ovens to industrial process control, medical instrumentation, decision-support systems, and portfolio selection. To understand why use of fuzzy logic has grown, we must first understand what is meant by fuzzy logic.

Fuzzy logic has two different meanings. In a narrow sense, fuzzy logic is a logical system, which is an extension of multi valued logic. However, in a wider sense fuzzy logic (FL) is almost synonymous with the theory of fuzzy sets, a theory which relates to classes of objects with un-sharp boundaries in which membership is a matter of degree. In this perspective, fuzzy logic in its narrow sense is a branch of FL. Even in its more narrow definition, fuzzy logic differs both in concept and substance from traditional multi valued logical systems.

The basic ideas underlying FL are explained very clearly and insightfully in Foundations of Fuzzy Logic. What might be added is that the basic concept underlying FL is that of a

linguistic variable, that is, a variable whose values are words rather than numbers. In effect, much of FL may be viewed as a methodology for computing with words rather than numbers. Although words are inherently less precise than numbers, their use is closer to human intuition. Furthermore, computing with words exploits the tolerance for imprecision and thereby lowers the cost of solution.

Another basic concept in FL, which plays a central role in most of its applications, is that of a fuzzy if-then rule or, simply, fuzzy rule. Although rule-based systems have a long history of use in Artificial Intelligence (AI), what is missing in such systems is a mechanism for dealing with fuzzy consequents and fuzzy antecedents. In fuzzy logic, this mechanism is provided by the calculus of fuzzy rules. The calculus of fuzzy rules serves as a basis for what might be called the Fuzzy Dependency and Command Language (FDCL). Although FDCL is not used explicitly in the toolbox, it is effectively one of its principal constituents. In most of the applications of fuzzy logic, a fuzzy logic solution is, in reality, a translation of a human solution into FDCL.[15,16]

A trend that is growing in visibility relates to the use of fuzzy logic in combination with neurocomputing and genetic algorithms. More generally, fuzzy logic, neurocomputing, and genetic algorithms may be viewed as the principal constituents of what might be called soft computing. Unlike the traditional, hard computing, soft computing accommodates the imprecision of the real world. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution cost. In the future, soft computing could play an increasingly important role in the conception and design of systems whose MIQ (Machine IQ) is much higher than that of systems designed by conventional methods. Among various combinations of methodologies in soft computing, the one that has highest visibility at this juncture is that of fuzzy logic and neurocomputing, leading to neuro-fuzzy systems. Within fuzzy logic, such systems play a particularly important role in the induction of rules from observations. Fuzzy logic is all about the relative importance of precision: How important is it to be exactly right when a rough answer will do.

Fuzzy control system design essentially amounts to

(1) Choosing the fuzzy controller inputs and outputs,

(2) Choosing the preprocessing that is needed for the controller inputs and possibly post processing that is needed for the outputs, and

(3) Designing each of the four components of the fuzzy controller.

There are standard choices for the fuzzification and defuzzification interfaces. Moreover an inference mechanism also settles and may use this for many different processes. Hence, the main part of the fuzzy controller that we focus on for design is the rule-base. The rule-base is constructed so that it represents a human expert “in-the-loop. Hence, the information that we load into the rules in the rule-base may come from an actual human expert who has spent long time learning how best to control the process. In other situations there is no such human expert, and the control engineer will simply study the plant dynamics (perhaps using modeling and simulation) and write down a set of control rules that makes sense. As an example, experience driving a car can practice regulating the speed about a desired set-point and load this information into a rule-base. For instance, one rule that a human driver may use is “If the speed is lower than the set-point, then press down further on the accelerator pedal.” A rule that would represent even more detailed information about how to regulate the speed would be “If the speed is lower than the set-point AND the speed is approaching the set-point very fast, and then release the accelerator pedal by a small amount.” This second rule characterizes our knowledge about how to make sure that we do not overshoot our desired goal (the set-point speed). Generally speaking, if we load very detailed expertise into the rule-base, we enhance our chances of obtaining better performance.

FL incorporates a simple, rule-based IF X AND Y THEN Z approach to a solving control problem rather than attempting to model a system mathematically. The FL model is empirically-based, relying on an operator's experience rather than their technical understanding of the system. For example, rather than dealing with temperature control in terms such as "SP =500F", "T <1000F", or "210C <TEMP <220C", terms like "IF (process is too cool) AND (process is getting colder) THEN (add heat to the process)" or "IF (process is too hot) AND (process is heating rapidly) THEN (cool the process quickly)" are used. These terms are imprecise and yet very descriptive of what must actually happen. Consider what you do in the shower if the temperature is too cold: you will make the water comfortable very quickly with little trouble. FL is capable of mimicking this type of behavior but at very high rate.[17]

FL requires some numerical parameters in order to operate such as what is considered significant error and significant rate-of-change-of-error, but exact values of these numbers are usually not critical unless very responsive performance is required in which case empirical tuning would determine them. For example, a simple temperature control system could use a single temperature feedback sensor whose data is subtracted from the command signal to compute "error" and then time-differentiated to yield the error slope or rate-of-change-of-error, hereafter called "error-dot". Error might have units of degs F and a small error considered to be 2F while a large error is 5F. The "error-dot" might then have units of degs/min with a small error-dot being 5F/min and a large one being 15F/min. These values don't have to be symmetrical and can be "tweaked" once the system is operating in order to optimize performance. Generally, FL is so forgiving that the system will probably work the first time without any tweaking.

**2.2 Features**

Fuzzy logic offers several unique features that make it a particularly good choice for many control problems. It is inherently robust since it does not require precise, noise-free inputs and can be programmed to fail safely if a feedback sensor quits or is destroyed. The output control is a smooth control function despite a wide range of input variations. Since the Fuzzy logic controller processes user-defined rules governing the target control system, it can be modified and tweaked easily to improve or drastically alter system performance. New sensors can easily be incorporated into the system simply by generating appropriate governing rules. Fuzzy logic is not limited to a few feedback inputs and one or two control outputs, nor is it necessary to measure or compute rate-of-change parameters in order for it to be implemented. Any sensor data that provides some indication of a system's actions and reactions is sufficient. This allows the sensors to be inexpensive and imprecise thus keeping the overall system cost and complexity low. Because of the rule-based operation, any reasonable number of inputs can be processed (1-8 or more) and numerous outputs (1-4 or more) generated. Fuzzy logic can control nonlinear systems that would be difficult or impossible to model mathematically.[18]

Fuzzy Logic is a paradigm for an alternative design methodology which can be applied in developing both linear and non-linear systems for embedded control. By using fuzzy logic, designers can realize lower development costs, superior features, and better end product performance. Furthermore, products can be brought to market faster and more

cost-effectively. It is An Alternative Design Methodology Which Is Simpler, And Faster. Some feature of fuzzy logic are as:

- **Fuzzy Logic reduces the design development cycle:** With a fuzzy logic design methodology some time consuming steps are eliminated. Moreover, during the debugging and tuning cycle you can change your system by simply modifying rules, instead of redesigning the controller. In addition, since fuzzy is rule based, you do not need to be an expert in a high or low level language which helps you focus more on your application instead of programming. As a result, Fuzzy Logic substantially reduces the overall development cycle.
- **Fuzzy Logic simplifies design complexity:** Fuzzy logic lets you describe complex systems using your knowledge and experience in simple English-like rules. It does not require any system modeling or complex math equations governing the relationship between inputs and outputs. Fuzzy rules are very easy to learn and use, even by non-experts. It typically takes only a few rules to describe systems that may require several of lines of conventional software. As a result, Fuzzy Logic significantly simplifies design complexity

- **Fuzzy Logic improves time to market:** Commercial applications in embedded control require a significant development effort a majority of which is spent on the software portion of the project. Development time is a function of design complexity, and the number of iterations required in a debugging and tuning cycle. A fuzzy based design methodology addresses both issues very effectively. Moreover, due to its simplicity the description of a fuzzy controller not only is transportable across design teams, but also provides a superior media to preserve, maintain, and upgrade intellectual property. As a result, Fuzzy Logic can dramatically improve time to market.
- **A Better Alternative Solution To Non-Linear Control:** Most real life physical systems are actually non-linear systems. Conventional design approaches use different approximation methods to handle non-linearity. Some typical choices are, linear, piecewise linear, and lookup table approximations to trade off factors of complexity, cost, and system performance. A linear approximation technique is relatively simple, however it tends to limit control performance and may be costly to implement in certain applications. A piecewise linear technique works better, although it is tedious

to implement because it often requires the design of several linear controllers. A lookup table technique may help improve control performance, but it is difficult to debug and tune. Furthermore in complex systems where multiple inputs exist, a lookup table may be impractical or very costly to implement due to its large memory requirements. Fuzzy logic provides an alternative solution to non-linear control because it is closer to the real world. Non-linearity is handled by rules, membership functions, and the inference process which results in improved performance, simpler implementation, and reduced design costs.[19]

- **Fuzzy Logic improves control performance:** In many applications Fuzzy Logic can result in better control performance than linear, piecewise linear, or lookup table techniques. With fuzzy logic we can use rules and membership functions to approximate any continuous function to any degree of precision.
- **Fuzzy Logic simplifies implementation:** The one input temperature controller presented so far has helped us illustrate some fundamental concepts, however real life control is much more complex in nature. Most control applications have multiple inputs and require modeling and tuning of a large number of parameters which makes implementation very tedious and time consuming. Fuzzy rules can help you simplify implementation by combining multiple inputs into single if-then statements while still handling non-linearity.
- **Fuzzy Logic reduces hardware costs:** Using a lookup table the two-input temperature controller requires 64 Kb of memory, while the fuzzy approach is accomplished with less than 0.5 Kb of memory for labels and object code combined. This difference in memory savings implies a cheaper hardware implementation. In addition, conventional techniques in most real life applications require complex mathematical analysis and modeling, floating point algorithms, and complex branching. This typically yields a substantial size of object code which requires a high end DSP chip to run. Fuzzy Logic enables you to use a simple rule based approach which offers significant cost savings, both in memory and processor class.[20]

### 2.3 The Rule Based Matrix

The fuzzy parameters of error (command-feedback) and error-dot (rate-of-change-of-error) were modified by the adjectives "negative", "zero", and "positive". To picture this, imagine the simplest practical implementation, a 3-by-3 matrix. The columns represent

"negative error", "zero error", and "positive error" inputs from left to right. The rows represent "negative", "zero", and "positive" "error-dot" input from top to bottom. This planar construct is called a rule matrix. It has two input conditions, "error" and "error-dot", and one output response conclusion (at the intersection of each row and column). In this case there are nine possible logical product (AND) output response conclusions.

In 1973, Professor Lotfi Zadeh proposed the concept of linguistic or "fuzzy" variables. Think of them as linguistic objects or words, rather than numbers. The sensor input is a noun, e.g. "temperature", "displacement", "velocity", "flow", "pressure", etc. Since error is just the difference, it can be thought of the same way. The fuzzy variables themselves are adjectives that modify the variable (e.g. "large positive" error, "small positive" error, “zero" error, "small negative" error, and "large negative" error). As a minimum, one could simply have "positive", "zero", and "negative" variables for each of the parameters. Additional ranges such as "very large" and "very small" could also be added to extend the responsiveness to exceptional or very nonlinear conditions, but aren't necessary in a basic system.

Although not absolutely necessary, rule matrices usually have an odd number of rows and columns to accommodate a "zero" center row and column region. This may not be needed as long as the functions on either side of the center overlap somewhat and continuous dithering of the output is acceptable since the "zero" regions correspond to "no change" output responses the lack of this region will cause the system to continually hunt for "zero". It is also possible to have a different number of rows than columns. This occurs when numerous degrees of inputs are needed. The maximum number of possible rules is simply the product of the number of rows and columns, but definition of all of these rules may not be necessary since some input conditions may never occur in practical operation. The primary objective of this construct is to map out the universe of possible inputs while keeping the system sufficiently under control.

### 2.4 Membership Functions

Membership functions are the building blocks of fuzzy set theory. A membership function is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1. The input space is sometimes referred to as the universe of discourse, a fancy name for a simple concept.

One of the most commonly used examples of a fuzzy set is the set of tall people. In this case, the universe of discourse is all potential heights, say from 3 feet to 9 feet, and the word tall would correspond to a curve that defines the degree to which any person is tall. If the set of tall people is given the well-defined (crisp) boundary of a classical set, you might say all people taller than 6 feet are officially considered tall. However, such a distinction is clearly absurd. It may make sense to consider the set of all real numbers greater than 6 because numbers belong on an abstract plane, but when we want to talk about real people, it is unreasonable to call one person short and another one tall when they differ in height by the width of a hair.[17]

Fuzzy membership functions determined subjectively in practical problems based on an expert's opinion. In such a situation one can think of membership functions as a technique to formalize empirical problem solving that is based on experience rather than the knowledge of theory. The expert's way of thinking can be captured either directly or through a special algorithm. Such determination could become more focused by physical measurements if the need arises. Available frequency histograms and other probability data can also help in constructing the membership function. It is important, however, to note that membership function values, or grades of membership, are not probabilities. Membership construction can be further simplified by selecting their form from the smaller family of the commonly used ones. So the membership function involves:

- Fuzzy sets describe vague concepts (e.g., fast runner, hot weather, weekend days).
- A fuzzy set admits the possibility of partial membership in it. (e.g., Friday is sort of a weekend day, the weather is rather hot).
- The degree an object belongs to a fuzzy set is denoted by a membership value between 0 and 1. (e.g., Friday is a weekend day to the degree 0.8).
- A membership function associated with a given fuzzy set maps an input value to its appropriate membership value.

### 2.5 If-Then Rules

Fuzzy sets and fuzzy operators are the subjects and verbs of fuzzy logic. These if-then rule statements are used to formulate the conditional statements that comprise fuzzy logic.

A single fuzzy if-then rule assumes the form

if $x$ is $A$ then $y$ is $B$

where $A$ and $B$ are linguistic values defined by fuzzy sets on the ranges (universes of discourse) X and Y, respectively. The if-part of the rule "$x$ is $A$" is called the *antecedent* or premise, while the then-part of the rule "$y$ is $B$" is called the *consequent* or conclusion. An example of such a rule might be

If service is good then tip is average

The concept good is represented as a number between 0 and 1, and so the antecedent is an interpretation that returns a single number between 0 and 1. Conversely, *average* is represented as a fuzzy set, and so the consequent is an assignment that assigns the entire fuzzy set B to the output variable $y$. In the if-then rule, the word *is* gets used in two entirely different ways depending on whether it appears in the antecedent or the consequent. In MATLAB terms, this usage is the distinction between a relational test using "==" and a variable assignment using the "=" symbol. A less confusing way of writing the rule would be

If service = good then tip = average

In general, the input to an if-then rule is the current value for the input variable (in this case, service) and the output is an entire fuzzy set (in this case, average). This set will later be defuzzified, assigning one value to the output. The concept of defuzzification is described in the next section.

Interpreting an if-then rule involves distinct parts: first evaluating the antecedent (which involves fuzzifying the input and applying any necessary *fuzzy operators*) and second applying that result to the consequent (known as *implication*). In the case of two-valued or binary logic, if-then rules do not present much difficulty. If the premise is true, then the conclusion is true. If you relax the restrictions of two-valued logic and let the antecedent be a fuzzy statement, how does this reflect on the conclusion? The answer is a simple one. if the antecedent is true to some degree of membership, then the consequent is also true to that same degree. Thus:

in binary logic: $p \rightarrow q$ ($p$ and $q$ are either both true or both false.)
in fuzzy logic: $0.5\,p \rightarrow 0.5\,q$ (Partial antecedents provide partial implication.)

The antecedent of a rule can have multiple parts.

: If sky is gray and wind is strong and barometer is falling, then

in which case all parts of the antecedent are calculated simultaneously and resolved to a single number using the logical operators described in the preceding section. The consequent of a rule can also have multiple parts.

: If temperature is cold then hot water valve is open and cold water valve is shut

all consequents are affected equally by the result of the antecedent. How is the consequent affected by the antecedent? The consequent specifies a fuzzy set be assigned to the output. The implication function then modifies that fuzzy set to the degree specified by the antecedent. The most common ways to modify the output fuzzy set are truncation using the main function or scaling using the prod function.

if-then rules is a three-part process.

1) **Fuzzify inputs**: Resolve all fuzzy statements in the antecedent to a degree of membership between 0 and 1. If there is only one part to the antecedent, then this is the degree of support for the rule.
2) **Apply fuzzy operator to multiple part antecedents**: If there are multiple parts to the antecedent, apply fuzzy logic operators and resolve the antecedent to a single number between 0 and 1. This is the degree of support for the rule.
3) **Apply implication method**: Use the degree of support for the entire rule to shape the output fuzzy set. The consequent of a fuzzy rule assigns an entire fuzzy set to the output. This fuzzy set is represented by a membership function that is chosen to indicate the qualities of the consequent. If the antecedent is only partially true, (i.e., is assigned a value less than 1), then the output fuzzy set is truncated according to the implication method.

In general, one rule alone is not effective. Two or more rules that can play off one another are needed. The output of each rule is a fuzzy set. The output fuzzy sets for each rule are then aggregated into a single output fuzzy set. Finally the resulting set is defuzzified, or resolved to a single number.[17,19.20]

## 2.6 Defuzzification

Defuzzification is an important operation in the theory of fuzzy sets. It transforms a fuzzy set information into a numeric data information. This operation along with the operation of fuzzification is critical to the design of fuzzy systems as both of these operations provide nexus between the fuzzy set domain and the real valued scalar domain. It is the process of producing a quantifiable result in fuzzy logic. Typically, a fuzzy system will have a number of rules that transform a number of variables into a "fuzzy" result, that is, the result is described in terms of membership in fuzzy sets. For example, rules designed to decide how much pressure to apply might result in "Decrease Pressure (15%), Maintain Pressure (34%), increase Pressure (72%)". Defuzzification would transform this result into a single number indicating the change in pressure. The simplest but least useful defuzzification method is to choose the set with the highest membership, in this case, "Increase Pressure" since it has a 72% membership, and ignore the others, and convert this 72% to some number. The problem with this approach is that it loses information. The rules that called for decreasing or maintaining pressure might as well have not been there in this case.

A useful defuzzification technique must first add the results of the rules together in some way. The most typical fuzzy set membership function has the graph of a triangle. Now, if this triangle were to be cut in a straight horizontal line somewhere between the top and the bottom, and the top portion were to be removed, the remaining portion forms a trapezoid. The first step of defuzzification typically "chops off" parts of the graphs to form trapezoids (or other shapes if the initial shapes were not triangles). For example, if the output has "Decrease Pressure (15%)", then this triangle will be cut 15% the way up from the bottom. In the most common technique, all of these trapezoids are then superimposed one upon another, forming a single geometric shape. Then, the centroid of this shape, called the fuzzy centroid, is calculated. The $x$ coordinate of the centroid is the defuzzified value.

## 2.7 How to Use

1) Define the control objectives and criteria: What am I trying to control? What do I have to do to control the system? What kind of response do I need? What are the possible (probable) system failure modes?

2) Determine the input and output relationships and choose a minimum number of variables for input to the FL engine (typically error and rate-of-change-of-error).

3) Using the rule-based structure of FL, break the control problem down into a series of IF X AND Y THEN Z rules that define the desired system output response for given system input conditions. The number and complexity of rules depends on the number of input parameters that are to be processed and the number fuzzy variables associated with each parameter. If possible, use at least one variable and its time derivative. Although it is possible to use a single, instantaneous error parameter without knowing its rate of change, this cripples the system's ability to minimize overshoot for a step inputs.

4) Create FL membership functions that define the meaning (values) of Input/output terms used in the rules.

5) Create the necessary pre- and post-processing FL routines if implementing in S/W, otherwise program the rules into the FL H/W engine.

6) Test the system, evaluate the results, tune the rules and membership functions, and retest until satisfactory results are obtained.

**2.8 Fuzzy Logic and Embedded Systems**

An objective of fuzzy logic has been to make computers think like people. Fuzzy logic can deal with the vagueness intrinsic to human thinking and natural language and recognizes that its nature is different from randomness. Using fuzzy logic algorithms could enable machines to understand and respond to vague human concepts such as hot, cold, large, small, etc. It also could provide a relatively simple approach to reach definite conclusions from imprecise information. Almost every application, including embedded control applications, could reap some benefits from fuzzy logic. Its incorporation in embedded systems could lead to enhanced performance, increased simplicity and productivity, reduced cost and time-to-market, along with other benefits. Fuzzy logic has the advantage of modeling complex, nonlinear problems linguistically rather than mathematically and using natural language processing (computing with words). The use of fuzzy logic requires, however, the knowledge of a human expert to create an algorithm that mimics his/her expertise and thinking. Also, studying the stability of a fuzzy system is a demanding task.

Numerous applications, including embedded ones, combine the use of fuzzy logic and neural networks. Neurofuzzy techniques take advantage of both fuzzy logic and neural networks, leading to systems that can:

- Mimic the human decision-making process
- Handle imprecise or vague information
- Learn by example and hence do not require the knowledge of a human expert
- Self-learn and self-organize.
- Process numeric, linguistic, or logical information

# Chapter 4: FPGA

## 3.1 Introduction

FPGA (Field Programmable Gate Array) is a type of integrated circuit (IC) containing a matrix of logic cells that can be programmed by a user to act as an arbitrary integrated circuit. For example, one can implement a processor, a digital filter or an interrupt controller on the basis of FPGA. Larger FPGAs even allows user to create complex system-on-chip (SoC) containing several interconnected components.

The process of FPGA engineering usually involves writing a description in a special language (hardware description language, HDL). The FPGA configuration data (firmware) is then created by special software from this description. Given the cost of an FPGA firmware development (which is relatively low) and the ability to reprogram FPGAs many times, it **4.2** isn't surprising that FPGAs has conquered a noticeable part of the market.

**3.2 ASIC vs. FPGA** The pros and cons of FPGAs versus ASICs are summarized as

| **FPGA advantages** | **ASIC advantages** |
|---|---|
| A. Low design costs, simplified design flow | A. Maximum performance for a given semico technology |
| B. Shorter time-to-market | B. Low production costs (for large volumes) |
| C. Reprogrammability | C. Low power consumption |

Consequently, FPGAs are used in low-volume products, where design costs comprise a significant part of the budget. On the contrary, ASIC design approach is often used for high-volume products, where high design costs (measured in millions of dollars) can be repaid with massive sales.[21]

## 3.3 History of Programming Logic

Prior to the invention of programmable logic electronic systems designers had to use specialized integrated circuits, each of which contained just a few gates. Such chips were called discreet

logic. In order to create even a moderately complex device one had to mount a few tens of chips on one board. This led to more complex board layout and reduced performance.

- Programmable Logic Arrays (PLAs): The first kind of programmable logic device (named Programmable Logic Array PLA*)* was introduced in early 70's. PLAs were one-time programmable chips containing AND and OR gates and able to implement a simple logic function represented as a disjunctive normal form. A PLA device can be defined by a three parameters:

1. Number of inputs,
2. Number of AND gates (terms),
3. Number of OR gates (= number of outputs).

There was also a variation of this architecture in which only the first commutation matrix (before AND gates) was programmable, and the second one was hardwired. These devices were called Programmable Array Logic.

Complex Programmable Logic Devices (CPLDs): Complex Programmable Logic Devices (CPLDs) chip includes logic blocks (microcells) at the borders of the chip, and a connection matrix located at the central part. Each macro cell has a structure similar to PLA. So, a CPLD device can be also seen as a set of PLAs on one chip with programmable interconnects. CPLDs are usually Flash-based, that is, the configuration of macro cells and the interconnection matrix is defined by contents of the on-chip Flash memory. It means that CPLD need not to be configured after each power-up, unlike the SRAM-based FPGAs. It must be noted that there are also Flash-based FPGAs. The main difference between CPLD and FPGA is not in configuration memory, but in the underlying architecture.

### 3.4 FPGA Architecture

Field Programmable Gate Array chips (FPGAs) are the most powerful and versatile programmable logic devices. They combine the idea of programmability from the earlier PLDs and the architecture of Uncommitted Gate Array which was one of the ways to develop ASICs. The main elements of an FPGA chip are a matrix of programmable logic blocks (these blocks are called differently, depending upon a vendor) and a programmable routing matrix.

FPGA configuration memory is usually based on volatile SRAM (static memory), so these FPGAs must be configured on every power-up from an external source, such as a dedicated Flash chip. However, there are also both SRAM-based FPGAs with integrated flash module and pure Flash-based FPGAs that don't use SRAM at all.

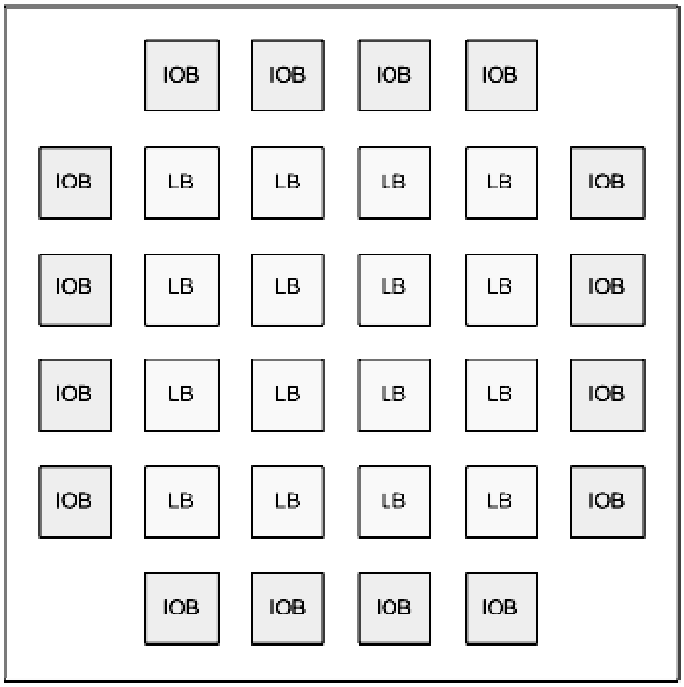


**Fig. 3: Logic Block (FPGA)**

FPGA Routing Blocks: A logic block usually contains LUTs (Look-Up Tables), also called function generators, and storage elements. LUTs are used for combinational logic implementation. For example, one 4-input LUT can implement any 4-input Boolean function. Storage elements can be configured either as edge-triggered flip-flops or as level-triggered latches. For example, typical Xilinx slice used in all Spartan FPGAs and all Virtex devices (except Virtex-5) includes two 4-input LUTs, two storage elements and dedicated arithmetic logic:

FPGA Routing Matrix: An FPGA device contains flexible programmable routing matrix which is used to connect logic blocks with each other. There are various type of connection lines in FPGA:

- Long lines are used to connect distant logic blocks,
- Short lines connect neighboring blocks with each other,
- Dedicated clock trees are used to distribute synchronization signals (these lines have large fan-out and little skew and jitter).
- Dedicated set/reset lines are used to reset all flip-flops in the FPGA.
- Other FPGA Resources: Modern FPGA resources aren't limited to just logic blocks and connection lines, and contain other useful elements:
- Block RAMs are dedicated memory blocks that can be used to implement in-chip data storage, FIFOs etc. Using in-chip memory blocks can eliminate the need to use external components.
- Clock management blocks are dedicated devices that can be used to synthesize various clock signals. They are frequently used when to raise FPGA internal performance (by raising its clock frequency) when it isn't desirable to increase board system frequency, which can lead to EMI (electromagnetic interference) problems.
- Dedicated DSP modules containing hardware MACs (Multiplier-Accumulators) that are building blocks for digital filters and other digital signal processing algorithms.
- Dedicated high-speed serial transceiver allows high-speed input/output. There transceivers support multiple serial communication standards.
- Dedicated hardware CPU. Although including dedicated hardware CPU cores in FPGA chip can be considered now somewhat old-fashioned, some relatively new Virtex-5 FX FPGAs from Xilinx contain a hard PowerPC core.

### 3.4.1 FPGA Configuration Memory

The configuration memory in FPGA devices can be organized differently.

- Most of the FPGAs are SRAM-based. They are volatile, that is, it is needed to configure them after each power-up. They usually have plenty of configuration modes.
- There are also nonvolatile SRAM-based FPGAs with integrated Flash. They are internally similar to plain SRAM-based FPGAs, the only difference being that they have internal Flash memory which can store configuration data. During the power-up the SRAM can be configured from the internal Flash, eliminating the need of external configuration memory.
- Flash-based FPGAs include ProASIC3 and Igloo families produced by Actel. Unlike the previous type, they don't contain configuration SRAM. Therefore, they consume much less

power than ordinary SRAM-based FPGAs. They are also considered to be more tolerant to radiation.

- Antifuse-based FPGAs include Actel*'s* Accelerator and RTAX families. They are the least vulnerable to radiation effects. Their main drawback is that they can only be programmed once.

### 3.4.2 FPGA Engineering Process

FPGA engineering process shown on next page usually involves the following stages:

1. Architecture design. This stage involves analysis of the project requirements, problem decomposition and functional simulation (if applicable). The output of this stage is a document which describes the future device architecture, structural blocks, their functions and interfaces.
2. HDL design entry. The device is described in a formal hardware description language (HDL). The most common HDLs are VHDL and Verilog.
3. Test environment design. This stage involves writing of test environments and behavioral models (when applicable). They are later used to ensure that the HDL description of a device is correct.
4. Behavioral simulation. This is an important stage that checks HDL correctness by comparing outputs of the HDL model and the behavioral model (being put in the same conditions).
5. Synthesis. This stage involves conversion of an HDL description to a so-called netlist which is basically a formally written digital circuit schematic. Synthesis is performed by a special software called synthesizer. For an HDL code that is correctly written and simulated, synthesis shouldn't be any problem. However, synthesis can reveal some problems and potential errors that can't be found using behavioral simulation, so, an FPGA engineer should pay attention to warnings produced by the synthesizer.
6. Implementation. A synthesizer-generated netlist is mapped onto particular device's internal structure. The main phase of the implementation stage is place and route or layout, which allocates FPGA resources (such as logic cells and connection wires). Then these configuration data are written to a special file by a program called bitstream generator.

7. Timing analysis. During the timing analysis special software checks whether the implemented design satisfies timing constraints (such as clock frequency) specified by the user.

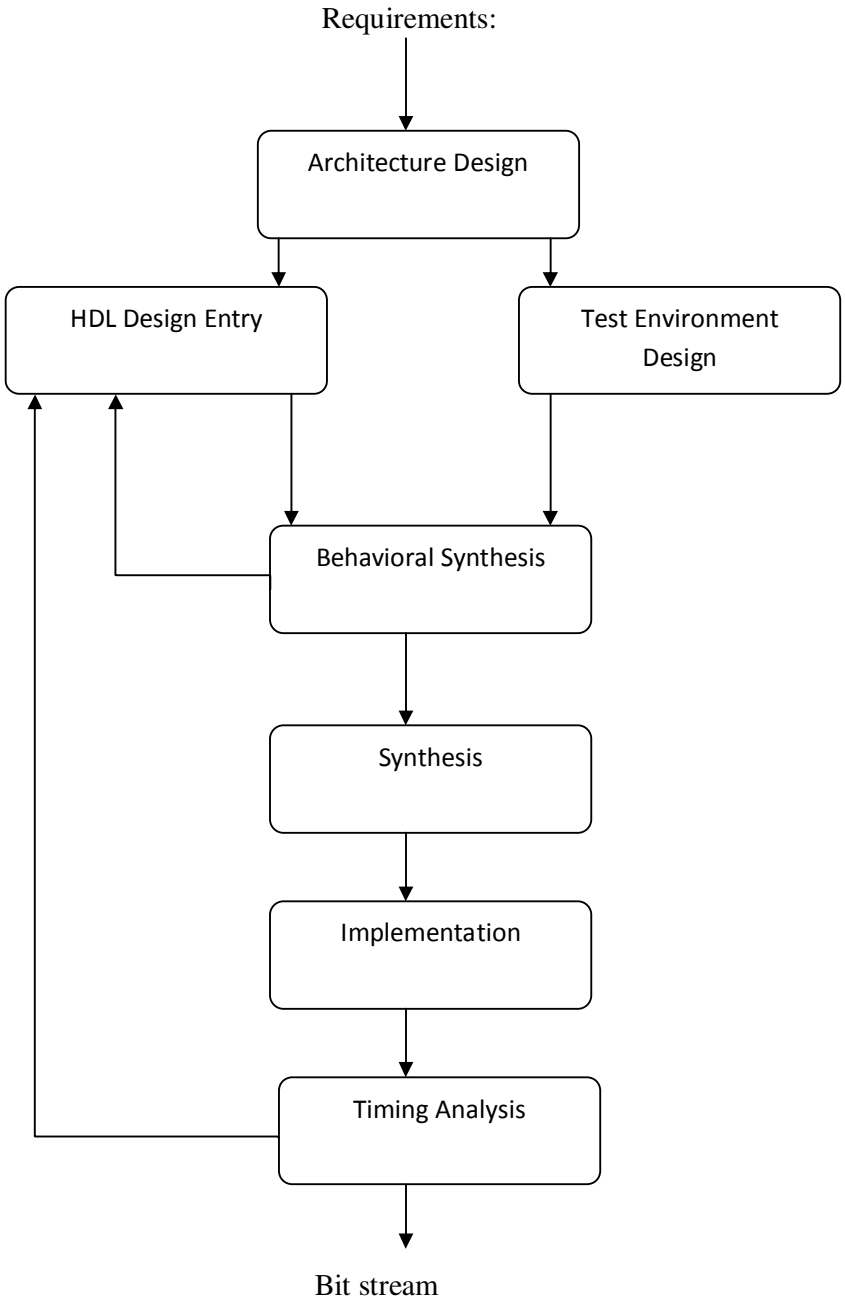


**Fig.4: Flow Chart representation of FPGA Engineering Process**

### 3.5 FPGA synthesis and implementation (Xilinx design flow)

This section describes FPGA synthesis and implementation stages typical for Xilinx design flow shown on Next page.

**Synthesis:** The synthesizer converts HDL (VHDL/Verilog) code into a gate-level netlist (represented in the terms of the UNISIM component library, a Xilinx library containing basic primitives). By default Xilinx ISE uses built-in synthesizer XST (Xilinx Synthesis Technology). Other synthesizers can also be used. Synthesis report contains many useful information. There is a maximum frequency estimate in the "timing summary" chapter. One should also pay attention to warnings since they can indicate hidden problems.

After a successful synthesis one can run "View RTL Schematic" task (RTL stands for register transfer level) to view a gate-level schematic produced by a synthesizer. XST output is stored in NGC format. Many third-party synthesizers (like Synplicity Simplify) use an industry-standard EDIF format to store netlist.

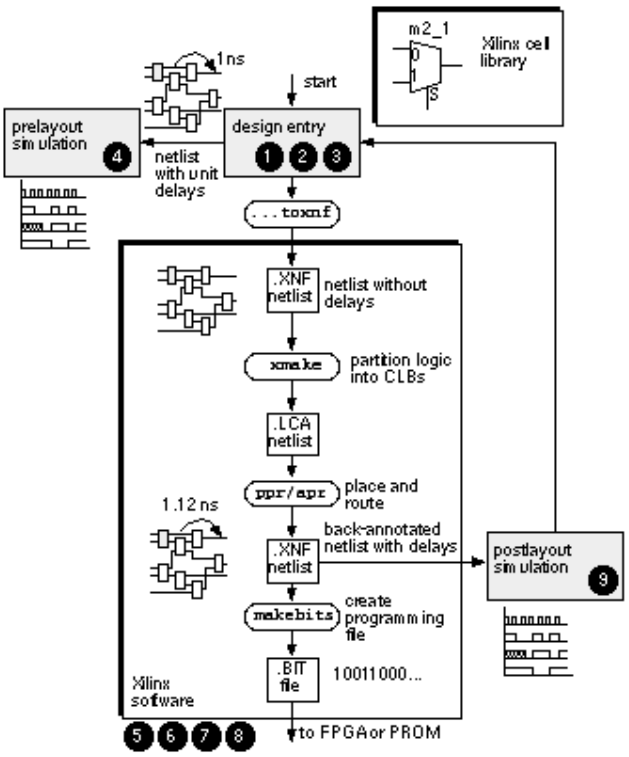


**Fig 5: Xilinx design flow (Source: Xilinx Inc.)**

**Implementation:** Implementation stage is intended to translate net list into the placed and routed FPGA design. Xilinx design flow has three implementation stages: translate, map and place and route. (These steps are specific for Xilinx: for example, Altera combines translate and map into one step executed by quartusmap.)

**Translate:** Translate is performed by the NGDBUILD program. During the translate phase an NGC netlist (or EDIF netlist, depending on what synthesizer was used) is converted to an NGD netlist. The difference between them is in that NGC netlist is based on the UNISIM component library, designed for behavioral simulation, and NGD netlist is based on the SIMPRIM library. The netlist produced by the NGDBUILD program contains some approximate information about switching delays.

**Map:** Mapping is performed by the MAP program. During the map phase the SIMPRIM primitives from an NGD netlist are mapped on specific device resources: LUTs, flip-flops, BRAMs and other. The output of the MAP program is stored in the NCD format. In contains precise information about switching delays, but no information about propagation delays (since the layout hasn't been processed yet. For Virtex-5 devices MAP also does placement (see below). For other devices placement is done by PAR. Routing is done by PAR for all devices.

**Place and route :** Placement and routing is performed by the PAR program. Place and route is the most important and time consuming step of the implementation. It defines how device resources are located and interconnected inside an FPGA. Placement is even more important than routing, because bad placement would make good routing impossible. In order to provide possibility for FPGA designers to tweak placement, PAR has a "starting cost table" option. PAR accounts for timing constraints set up by the FPGA designer. If at least one constraint can't be met, PAR returns an error. The output of the PAR program is also stored in the NCD format. For Virtex-5 devices, placement is performed by MAP instead of PAR.[22]

Timing constraints: In order to ensure that no timing violation (like period, setup or hold violation) will occur in the working design, timing constraints must be specified. Basic timing constraints that should be defined include frequency (period) specification and setup/hold times for input and output pads. The first is done with the PERIOD constraint, the second - with the OFFSET constraint. Timing constraints for the FPGA project are defined in the UCF file. Instead of editing the UCF file directly, an FPGA designer may prefer to use an appropriate GUI tool. However, the first approach is more powerful.

**I/O pads assignment:** I/O pads constraints should be specified in any real design. In the absence of these constraints the implementation tools will choose pads on its own. It is not that an FPGA designer usually need. I/O pad constraints can be conveniently set up with the Assign Package Pins utility from Xilinx ISE. The example of an UCF syntax for I/O pads constraints is:

NET "data_input" LOC = AK14 | IOSTANDARD = LVCMOS33 | SLEW = SLOW;

LOC constraint specifies a pin number (AK14). IOSTANDARD constraint specifies I/O standard (low-voltage CMOS 3.3V). SLEW constraint specifies slew rate (slope steepness). It is recommended to use SLOW slew rate whenever possible, since it reduces EMI (electromagnetic interference). There are also other parameters that can be set for I/O pads.

### 3.6 Programming SPARTAN 3 Board

Creating a new Project and Source

For WINDOWS operating system:

Start the Xilinx ISE project navigator by double clicking the Xilinx ISE icon on desktop.

OR

For LINUX operating system:

At TERMINAL write ise and press ENTER

PROJECT NAVIGATOR will be opened

**A.** Click on File and select New Project

Select a project location and type the name of the project eg.“VHDL_PROJECT”:

Click Next

Select the device family, device, package, and speed grade if required otherwise click on NEXT

Click Next

Click Next till we get FINISH

**B.** Click New Source

Select VHDL Module in the New Source Wizard window and name the file eg.”halfadder”

Click Next

Specify the inputs and outputs of design (HalfAdder).

Click Next

Click Finish if it satisfies the specifications

Click Next

Click Next

Click Finish.

Double-click on “HalfAdder.vhd” tab in the “Sources” panel

Complete the architectural part of the VHDL code.

Save it

**C.** Click New Source

Click Test bench Waveform

Name the test bench e.g. “halfaddertestbench”

Click Next

Click Finish.

Click Finish.

A waveform will be displayed. Click on the input and output waveforms

Click SAVE

**D.** Click and select “BEHAVIOURAL” in the top left “Sources” pane

Click waveform file “halfaddertestbench”

Click “Process” pane in the left of the window.

click on “+” sign “XILINX ISE SIMMULATOR” in the “Process” pane in the left window.

Double-click on “SIMMULATE BEHAVIOURAL MODEL” in that.

Finally a waveform will appear

Make sure the waveform satisfies the behavior of the entity

**E.** Click and select “SYNTHESIS” in the top left “Sources” pane

click on “+” sign “USER CONSTRAINTS” in the “Process” pane in the left of the window.

Double-click on “EDIT CONSTRAINTS” in that . a window will appear

Assign pin to each inputs and outputs as shown below…

NET ”A” LOC=”N17”|IOSTANARD=LVTTL;

NET ”b” LOC=”F12”|IOSTANARD=LVTTL;

NOTE1: Here N17 and F12 are pin no assigned to inputs a and b. Similarly assign pins to outputs also.

click on “+” sign “IMPLEMENT DESIGN” in the “Process” pane in the left of the window.

Double-click on “PLACE & ROUTE” in that a window will appear

Wait for the completion of the process

NOTE2: For windows pins are assigned as follows:

Double-click on “Assign Package Pins” in the “Process” pane in the left of the window.

You may be asked to save your VHDL code. Your design will be checked for syntax error. If you have any error, make sure you fix them before proceeding.

Click Yes.

Click Yes.

The Pace editor is loaded.

You can select “Package View” tab at the bottom of the right pane. The package view gives a better view of the physical FPGA package).Type in the desired pin names for each signal in the “Design Object List” at the left in the “Loc” column

Click File and Save.

Click File and Exit.

The following dialog may appear when saving the file:

Click on “Don’t show this dialog again”.

Click Ok.

View the UCF file by double-clicking “Edit Constraints (Text)” in the project Navigator window.

**Programming the Board**

In the Project Navigator window, double-click on “(Half Adder” tab in the “Sources” pane.

Right-click on “Generate Programming File” in the “Processes” pane.

Select “Properties”.

In the Process Properties windows, Select “Startup Options” tab.

Change “FPGA Start-UP Clock” to “JTAG Clock”

Click Apply. then Click Ok.

In the “Processes” window, click on the + sign by “Generate programming file”.

Double-click on “Configure Device (iMPACT)” or right click and click on RUN

Don’t close the opened window

It will automatically be closed and 2$^{nd}$ window will appear

Click Finish.

Select the .bit file

Click Open

Click Bypass

Click Bypass.

You will now be at the main iMPACT window:

Right-click on the FPGA (“xc3s200”).

Select “Program”.

Click Apply.

Click Ok.

A message will appear as “PROGRAM SUCCEEDED”

Now give the input to the board and check the outputs.

# Chapter 4: Result

## 4.1 System study

Traditionally, the operation and management of D.G. Sets in captive power plants is done manually which required human effort to decide the areas of the college campus where the power supply is provided during the time when main power is cutoff. Decision by a human being may be biased and so such inadequate decision not only leads to inefficient use of available capacitive power but also create unsatisfaction among its users. Moreover, extra labor is required to make the decision of the distribution of the power supply. Thus, an intelligent automated and unbiased system is required to make optimum use of the power in the most efficient manner.

This thesis emphasizes on the development of a Fuzzy Logic based optimal management of available power of 125kW from Diesel Generator (D.G.) set for residential college campus. The system designed in case of overload is capable of making intelligent decisions to make optimum use of power in the most efficient manner. The concept of Fuzzy Logic has been proved to be an effective way of improving a system operation, has been extended to this project.. During power cuts, the Campus is left with D.G. Set power of 125 kW to backup. For the undisturbed functioning of college and efficient use of remaining power, this FLC plays the role of automatic load shedding for specific area(s) in case of overload; otherwise there is no requirement of power cut(s).

The load shedding is done, based on the time and the load at that point of time sensed by clock and current transformer circuitry respectively. College campus is divided into various sections viz. academic area, staff residential area, street lights, hostels, office air conditioners, sewage treatment plant tube-well, etc. depending upon load distributions and FLC to take decisions for efficient and appropriate use of captive power among these sections for the smooth functioning of campus.

The block diagram of the system is shown on next page. Its input port having five inputs as:

a. Load sensed
b. Mains
c. Clock in
d. Reset
e. Select Lines

The output ports are as :

a. Air Conditioner
b. Academic Blocks
c. Hostel A
d. Hostel B
e. Residential Block
f. Display

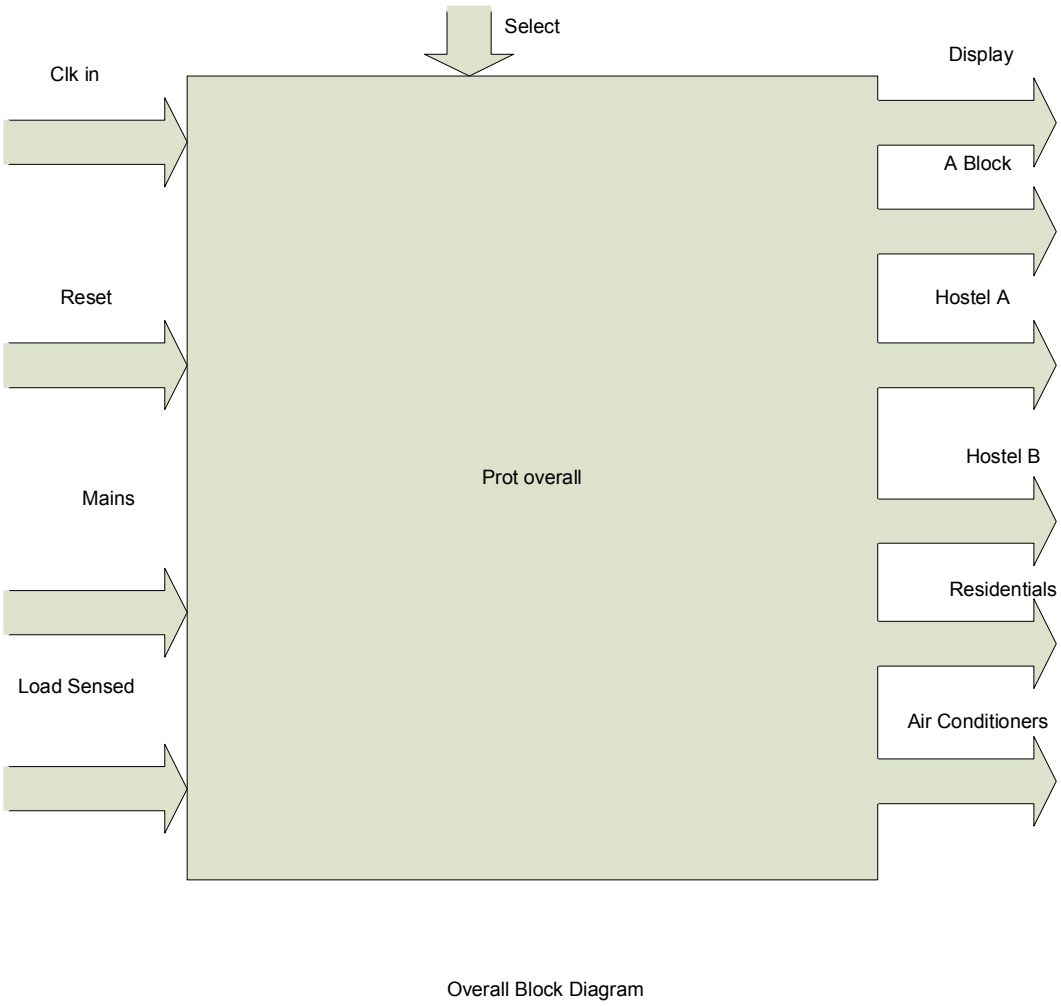


**Fig.6: Overall Block Diagram**

This fuzzy logic controller consists of six individual intricately designed modules includes

1. Timer Module : Generate clock of 100 MHz for Virtex 4 and synchronizes one pulse with one second.
2. Load Memory : Memory Unit for Load Memory stores the value of load for the most recent 24 Hrs.
3. Load Module : Load fuzzifier fuzzifies the Load input
4. Time Module : Time fuzzifier fuzzifies the Time input.
5. Rule Matrix Module: If –Else logic unit used to make the decision based upon the inputs.

6. Display Select Module : Seven Segment Display Unit which multiplexes the five fuzzified outputs and the clock for display on the seven segment display present on Spartan Kit.

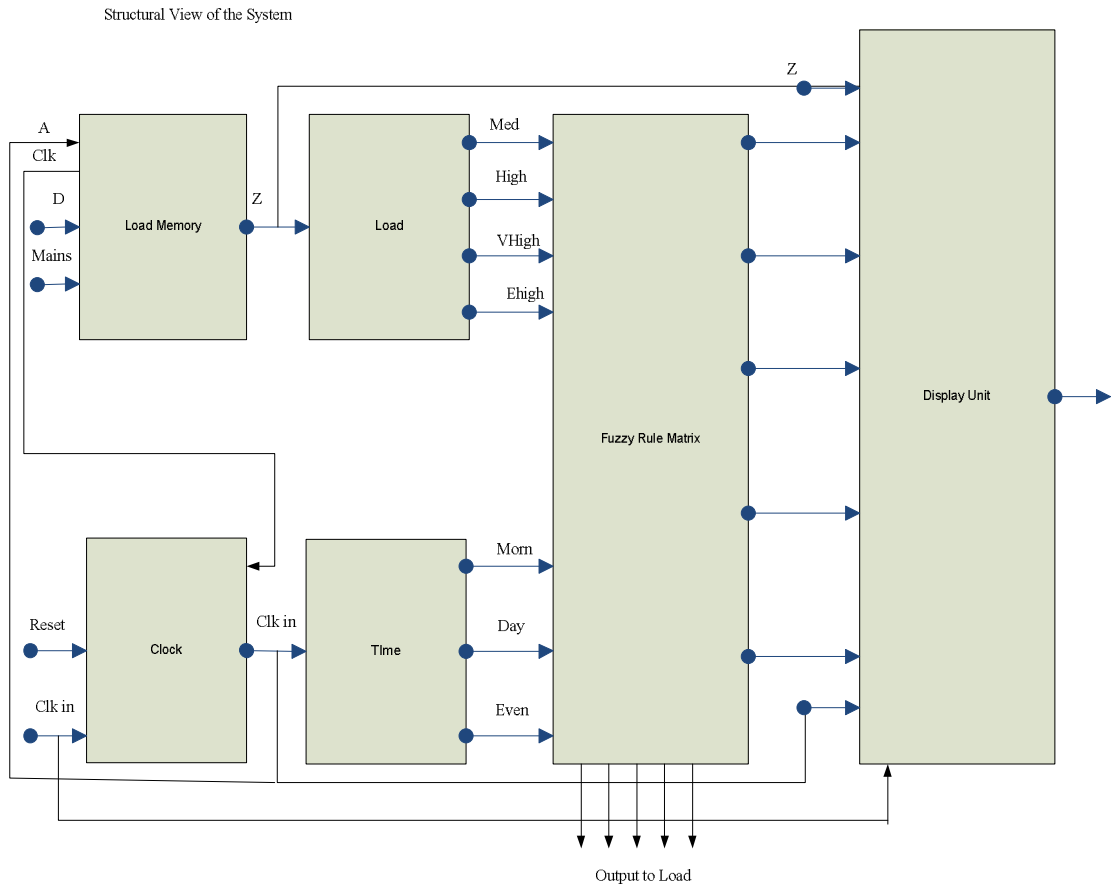


**Fig7. Structural view of the system**

### 4.2 Fuzzily Specified Timing & Load Conditions

The theoretical analysis of this system revealed that while dealing with load shedding, the operator is able to specify the 'Time' and 'Load' at best only fuzzily. This results in some amount of mismanagement of the areas to be served with power by definitive consequential user dissatisfaction. *TIME & LOAD*, being fuzzily specified, become the two input decision making variables for the FIS (Fuzzy Inference System). These are fuzzified into three and five fuzzy sets respectively. The *TIME* variable is fuzzified into three fuzzy sets: *MNight* (Morning Night), *Day* (Day) & *ENight* (Evening Night). Similarly, *LOAD* input variable is fuzzified into five appropriate fuzzy sets: *Low* (Low), *Medium* (Medium), *High* (High), *VHigh* (Very High) & *XHigh* (Extremely High). Of course, when *LOAD* will be *Low* no area should face power cut. Due to larger spans over the UOD (Universe of Discourse) between fuzzy sets of *TIME*, these are taken to be of

trapezoidal shapes while fuzzy sets for *LOAD* are taken to be of triangular shapes for reasons of economic representation. The output variable is taken as the command to relays controlling different sections.

A total of 125KW power of D.G. Set is to be managed to serve five areas and hence output OFF-AREA is divided into five fuzzy sets, viz. *ACs* (Office Air Conditioners), *HosA* (A-Group of two Hostels), *HosB* (B-Group of two Hostels), *Acad* (Academic Area) and *Resi* (Staff Residential Area). Peak loads of *ACs*, *HosA*, *HosB*, *Acad* and *Resi* are 40kW, 40kW, 40kW, 80kW, 70kW respectively. In this application, ANDing operation on antecedents and Sugeno-style of inference system is useful and the corresponding Fuzzy Rules Matrix is designed for suitable decisions to be taken on which areas are to be shed, as shown in Table 1.

| | | **TIME** | | |
|---|---|---|---|---|
| | | **MNight** | **Day** | **ENight** |
| **LOAD** | **Medium** | ACs | HosA | ACs |
| | **High** | Acs & Acad | ACs & HosA | ACs & HosB |
| | **VHigh** | Acs & Acad | ACs, HosA & HosB | ACs, HosA & HosB |
| | **XHigh** | ACs, Acad & Resi | ACs, HosA, HosB & Resi | ACs, HosA, HosB & Resi |

**Fig.8 Fuzzy Rule Matrix specifying OFF-AREA (for various Timings & Loads**).

The theoretical analysis of this problem revealed that while dealing with load shedding, operator can specify the TIME and LOAD are the parameters based on which decision has to be taken. Due to reason of economic representation of shapes of various fuzzy sets of TIME & LOAD are taken to be of either trapezoidal or triangular shapes shown in the Fig. 9. TIME is fuzzily specified in three trapezoidal fuzzy sets namely; MNight (Morning Night), Day (Day) & ENight (Evening Night).

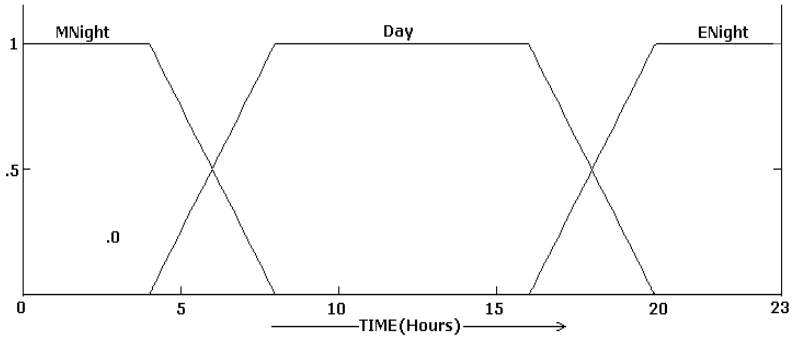


**Fig. 9 : Fuzzily specified *TIME***

In fig. 10,LOAD in five triangular fuzzy sets as Low (Low), Medium (Medium), High (Hight), VHigh (Very High) & XHigh (Extra High)

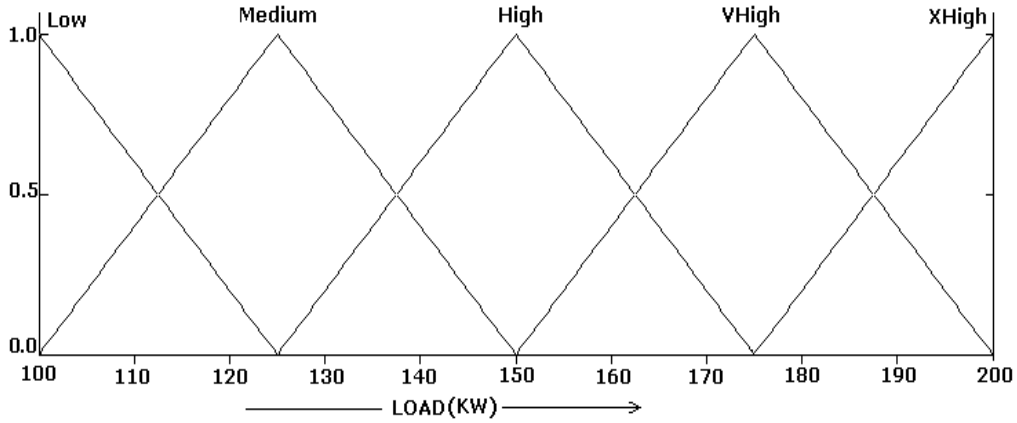


**Fig.10: Fuzzily specified LOAD**

### 4.3 Use of VHDL & Corresponding Block Diagram

As the system is designed for implementation on an FPGA chip, so the rule matrix is designed for the minimum possible rules. To that end, those rules which give similar output are aggregated using OR operation. VHDL is used for programming & implementation of this fuzzy system on Xilinx FPGA chip Vertex II Pro. Time & Load inputs are digitized into 8-bits & fuzzified, and upon rule firing, the digital outputs converted into analog to trigger the relays. If any output comes up with the net membership value equal to or greater than 50% of the maximum, then corresponding relay is triggered to switch off that area. The block diagram is shown below:

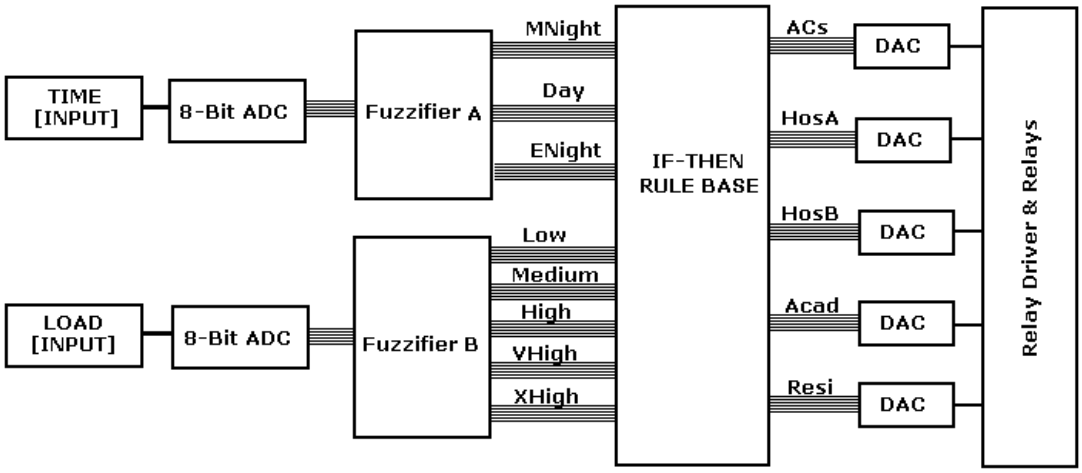


**Fig.11: System Block diagram.**

### 4.4 Details of Individual modules

#### 4.4.1 A Timer Module

A 8 bit counter output is delivered to 'Load LUT', as address bus and Time Fuzzifier Module as input . Universe of Discourse (UOD) of Time, 24 Hours, is hence quantized into 96 level each of 15 minutes. Hence the 8 bit counter for quarter hour pulses is reset when decimal count reaches 96. The block diagram of the timer module is as follow.

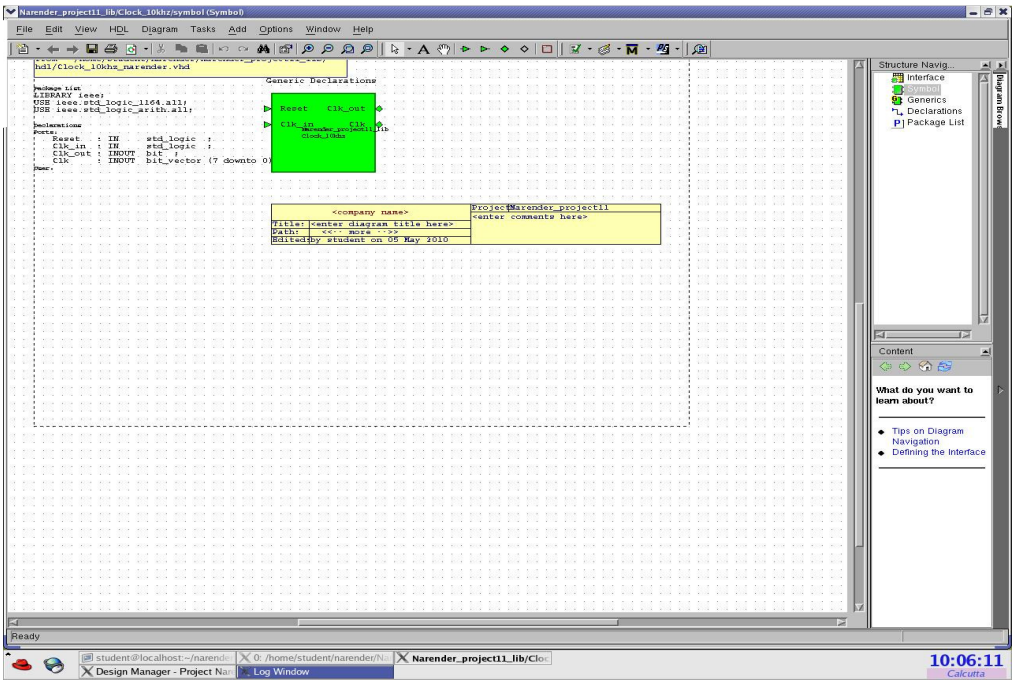


**Fig.12 : Block Diagram representation of Timer Module**

#### 4.4.2 Time Fuzzifier:

Output of the Digital Time Module is fed to the Time Fuzzifier module to fuzzify crisp 8 bit 'time' input into three fuzzy sets namely MNIGHT, DAY and ENIGHT for each of 8 bits respectively. These fuzzy sets are designed from the database of general opinion of a set of thirty people and are a bit modified to be of trapezoidal shapes, which are economic and simple to represent. An HDL code is generated for the Time Fuzzifier module.[23]

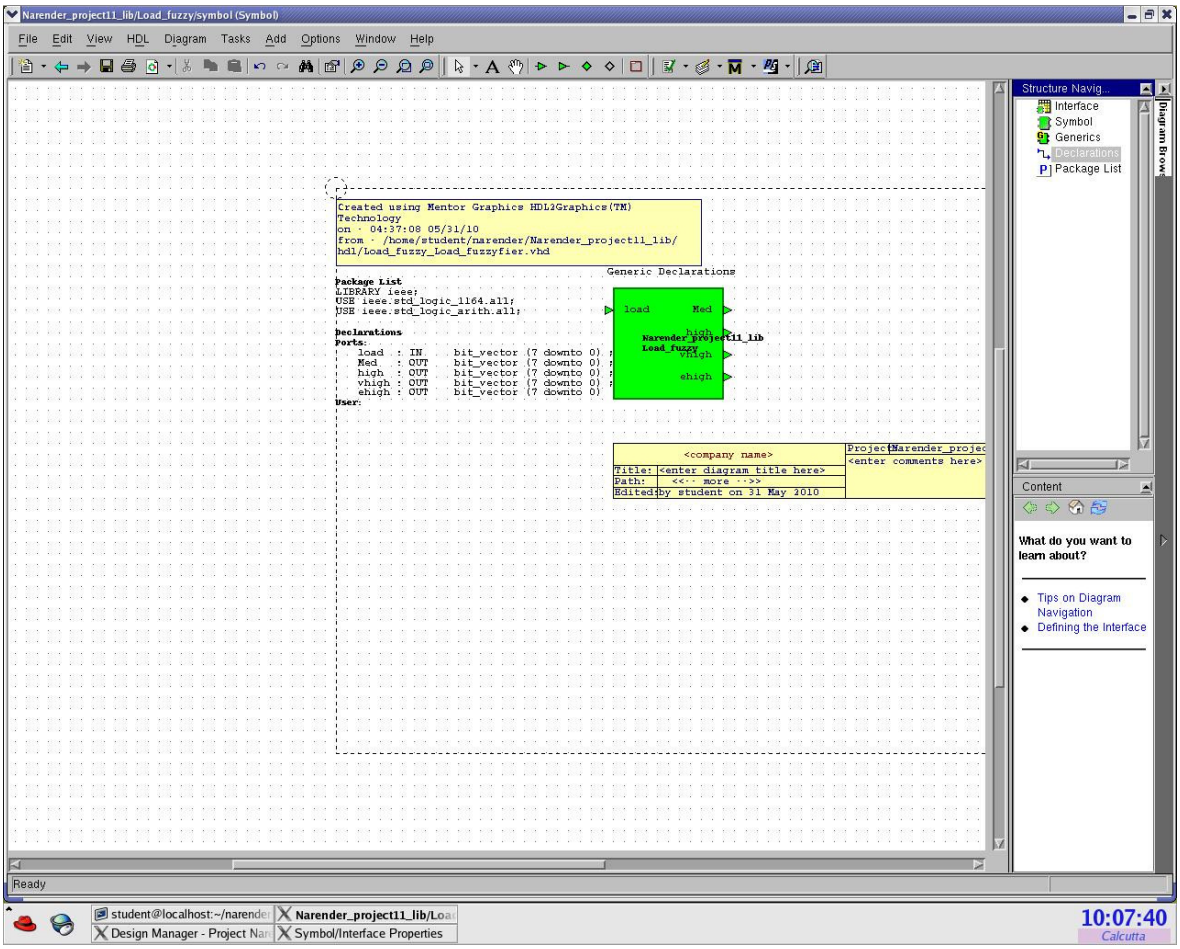


**Fig.13: Block Diagram representation of Time Fuzzifier Module**

### 4.4.3 Load Fuzzifier

Maximum capacity of the D.G. Set is 125 kW and the maximum load of the campus can be up to 200 kW. Hence for Load , input is taken to be from 100 kW to 200 kW range because load below 100 kW is definitely very low and can be served by the D.G. set without any problem. The load is divided into five fuzzy sets with overlaps, Low (Low), Medium (Medium), High (High), VHigh (Very High) and XHigh (Extra High).

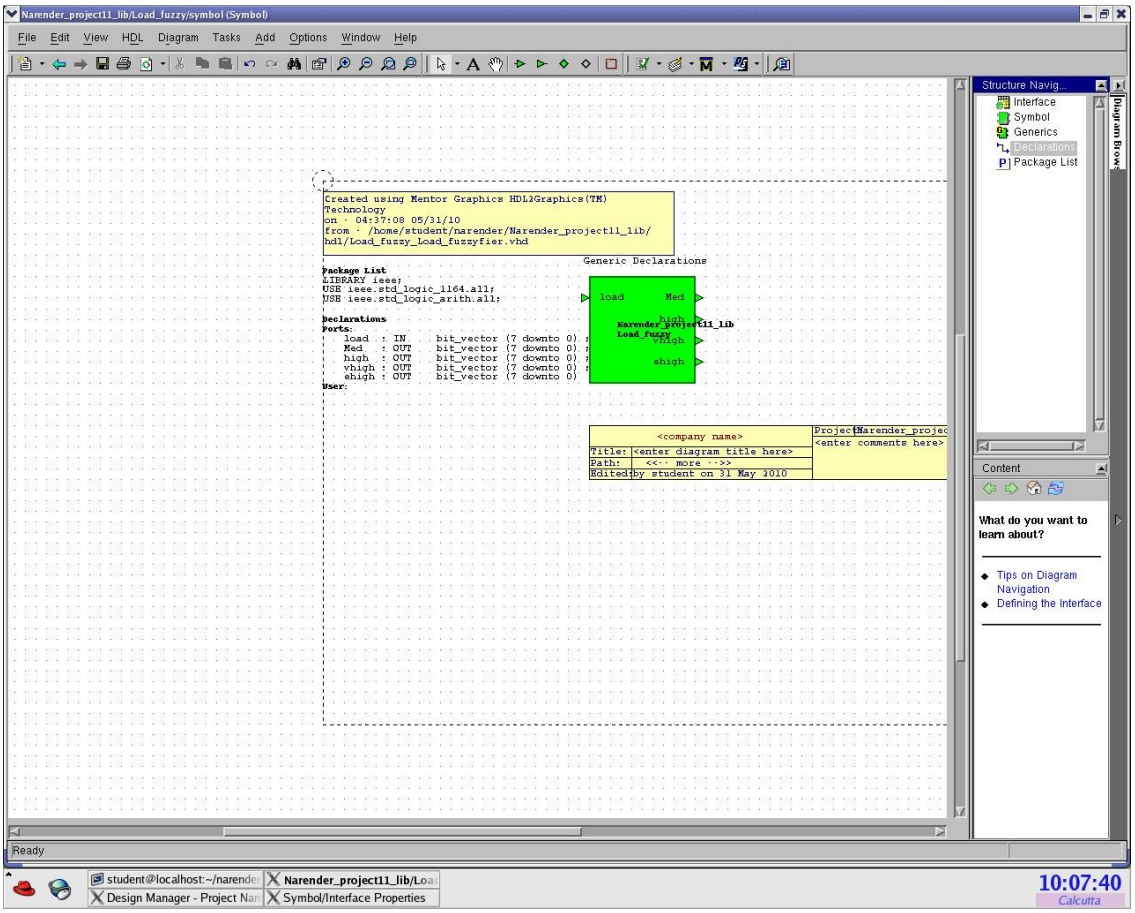


**Fig.14: Block Diagram representation of Load Fuzzifier Module**

The Load fuzzifier module accepts the 8-bit load input from the LOAD and delivers five out puts of 8 bit each , corresponding to each fuzzy set.

### 4.4.4 Rule Matrix

It uses five and three fuzzy sets respectively for load and time inputs which lead to a rule base of twelve minimum but adequate rules. Even a single fired rule may lead to switching off multiple sections if load exceeds the capacity of the D.G. set. If Load is less than the D.G. set capacity , then no section of the campus needs to be shed off, irrespective of the Time. So , during designing of the rule base , Low fuzzy Set is not included for making any decision.

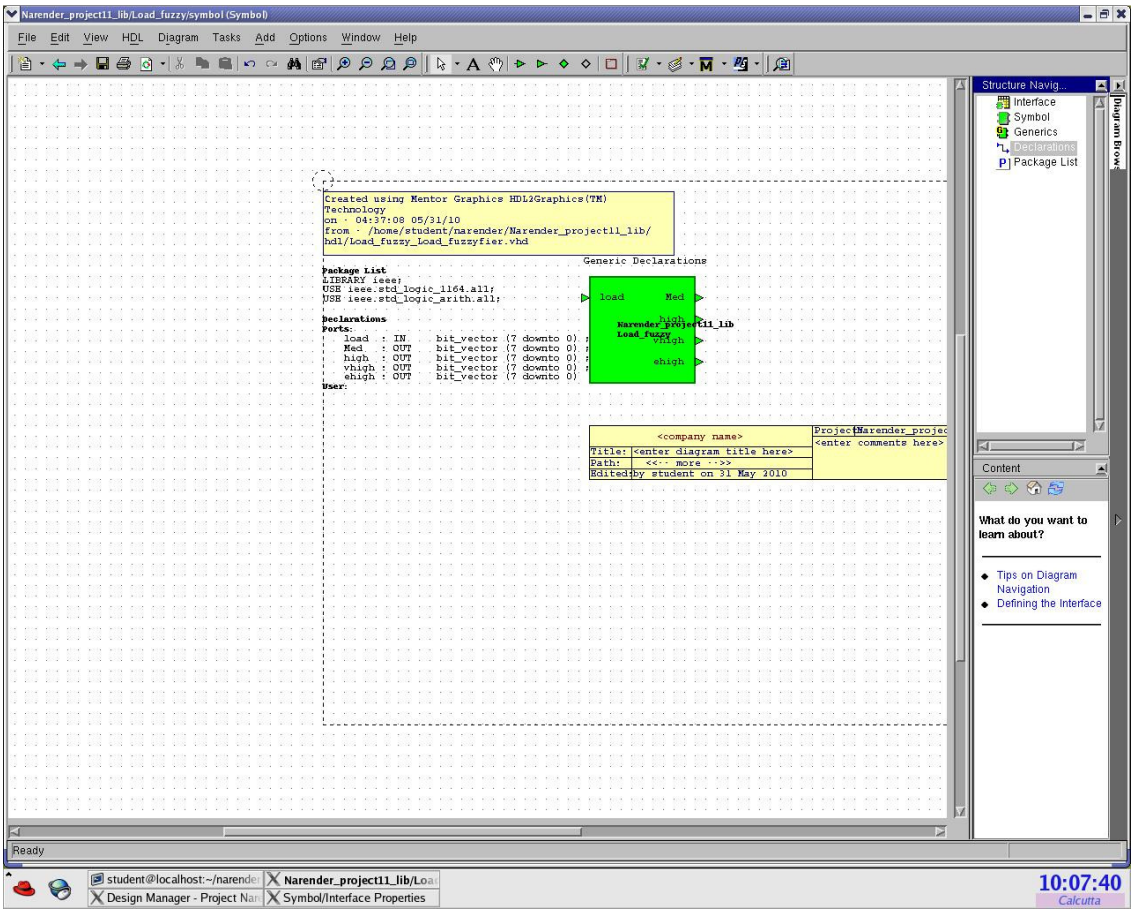


**Fig.15: Block Diagram representation of Rule Matrix Module**

### 4.4.5 Load Memory

We need to include a memory unit to store the data of Load of 24 hours when the mains are on. This is specially required in order to maintain its accuracy of functioning during long duration of power outage. Suppose power goes ff at 4 PM, which is considered to be day .so the D.G. set is will fire a set of rule for TIME=DAY and LOAD = XYZ.value. But if the power continue to be off till 8 PM, which is considered as EVENING NIGHT, so the D.G. set is required to fire new rules for TIME=EVEN_N and definitely needs to know the new set of value of Load at that particular point of time. Therefore we need a memory unit which will store the LOAD values for 24 hrs and will be refreshed every 15 second.

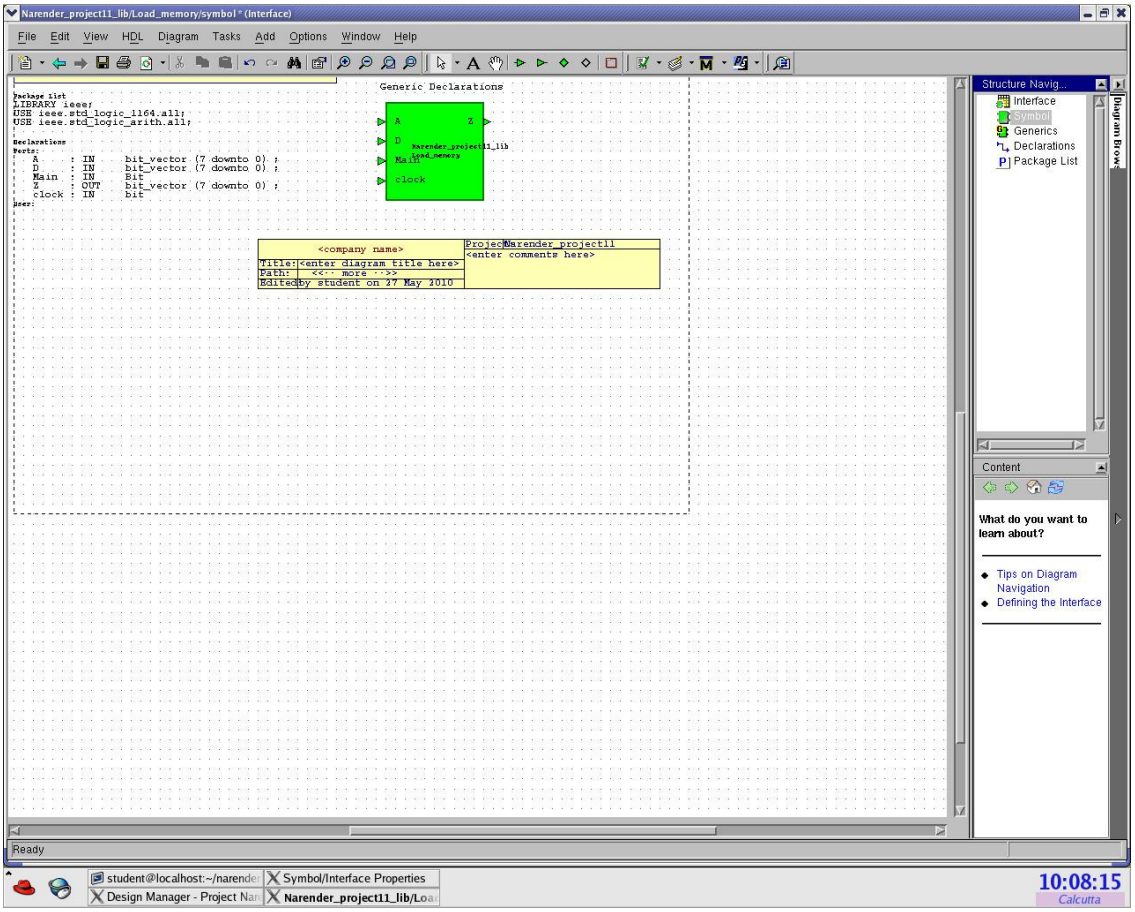


**Fig.16: Representation of Load Memory Module**

The above block diagram shows that the Load_Memory has 4 input ports and a single output port which will feed into LOAD unit .

### 4.4.6 Display Unit

A Display unit named display_select been designed to display various outputs. The display unit has 8 input ports and works as a multiplexer to display the specific input selected with the help of select line S (3 bits) otherwise if none of the input is elected , then it will , by default, display clock. The display device in case of SPARTAN 3 is a seven segment LED Display and in VERTEX 4 is 16 ×2 LCD display device. For example is s=011, then it will display the value of ac_disp on the output port.

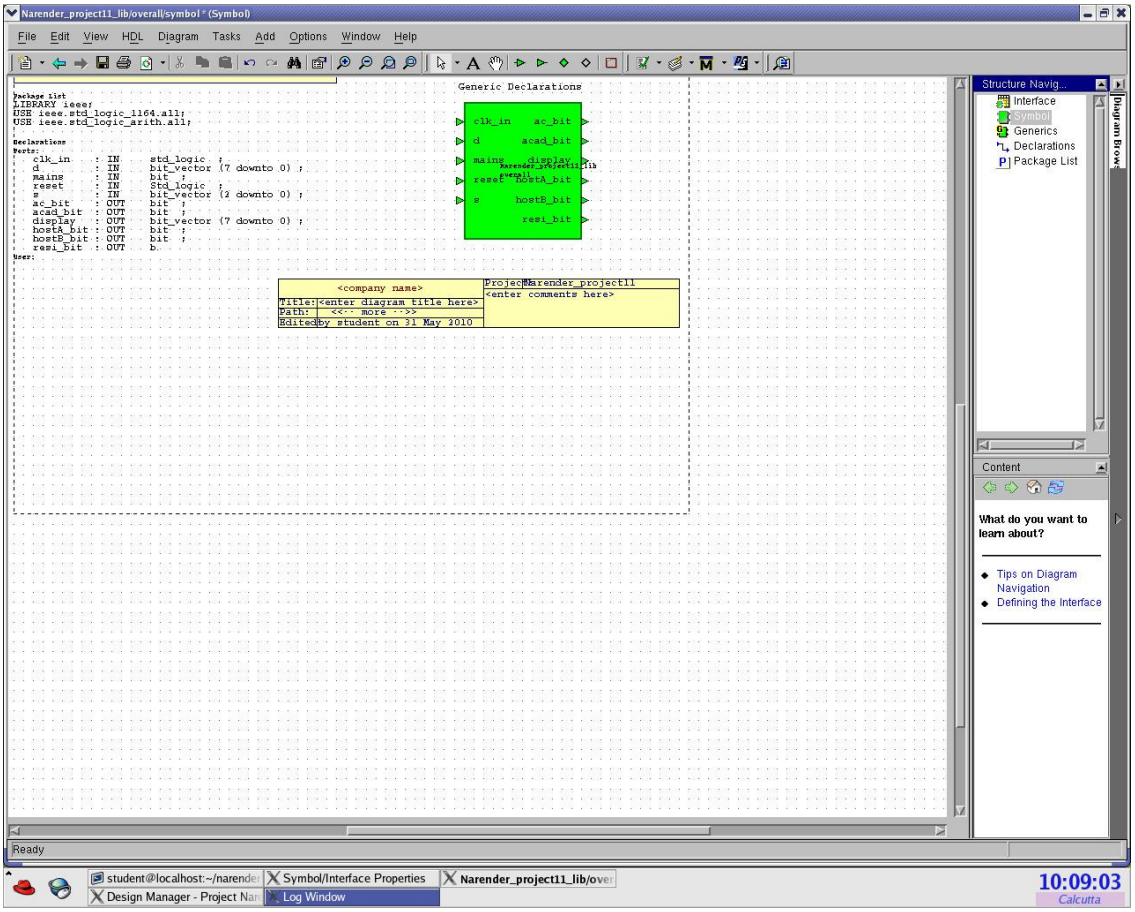


**Fig.17: Representation of Display Module**

## 4.5 Simulated Result of modules

Simulation of clock unit

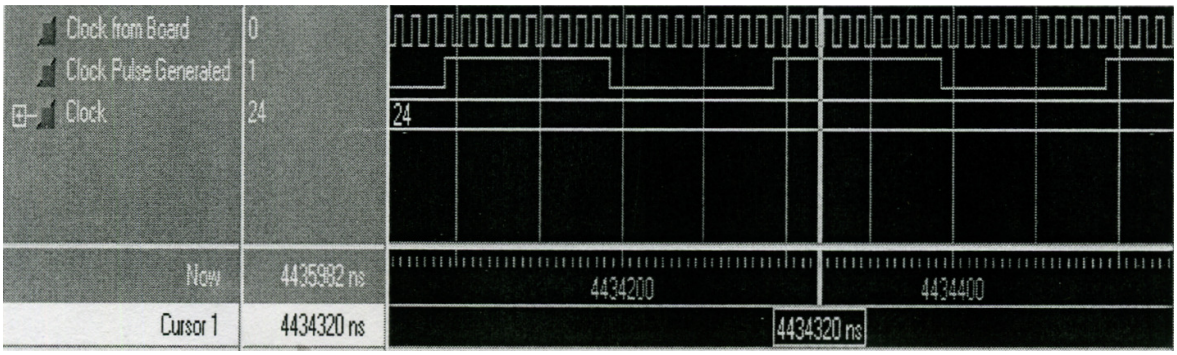


**Fig.18: Clock unit generates a pulse of 1 Second**

Simulation of Load_memory

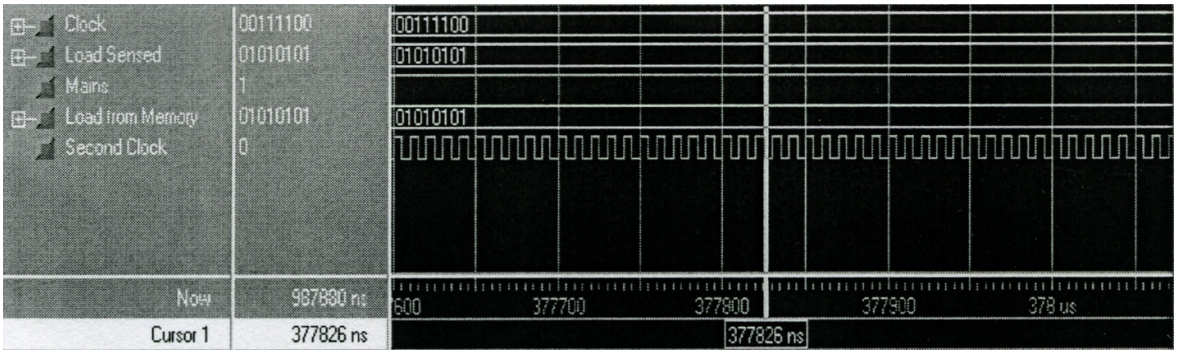


**Fig. 19: Simulation of Load Memory**

Clock (input port a) = 00111100, Load Sensed (input port d) =1010101, Mains=1 then load from memory=01010101, Mains = 1 then load from memory = 01010101

Simulation of Load Fuzzifier

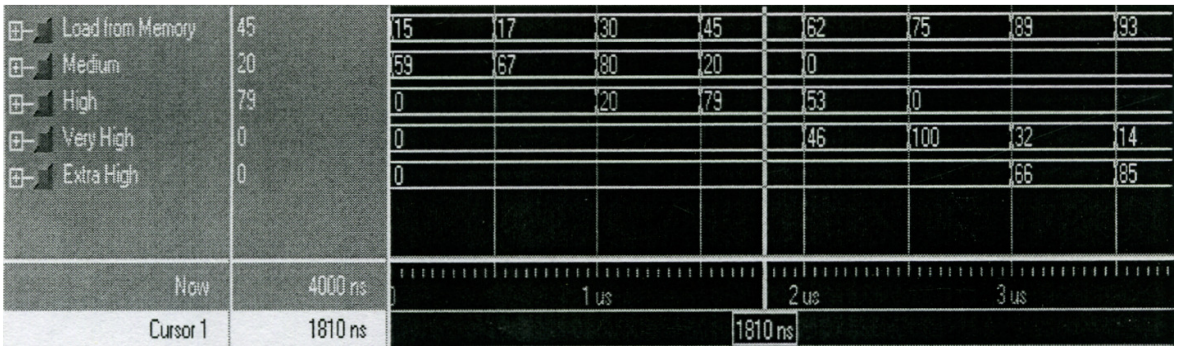


**Fig. 20: Simulation of Load Fuzzifier**

Load from Memory = 45, Fuzzified output =High

Simulation of Time Fuzzifier

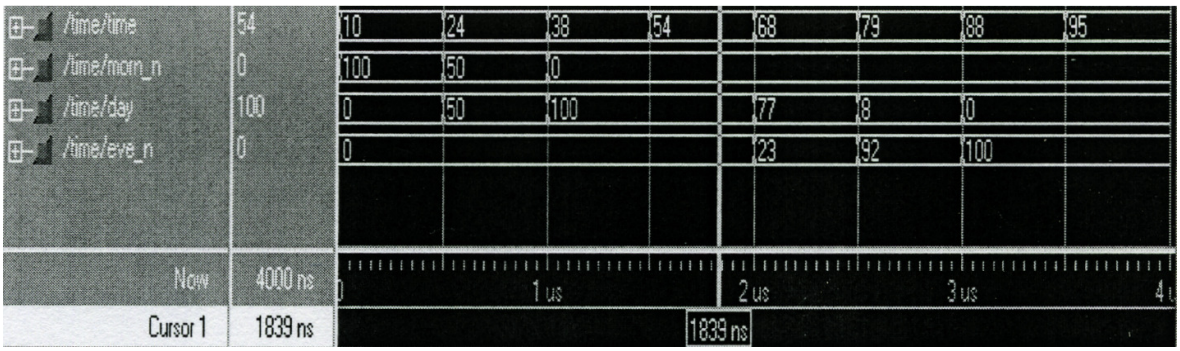


**Fig. 21: Simulation of Time Fuzzifier**

Time input = 54, Fuzzified output = Day

Simulation of Rule Matrix

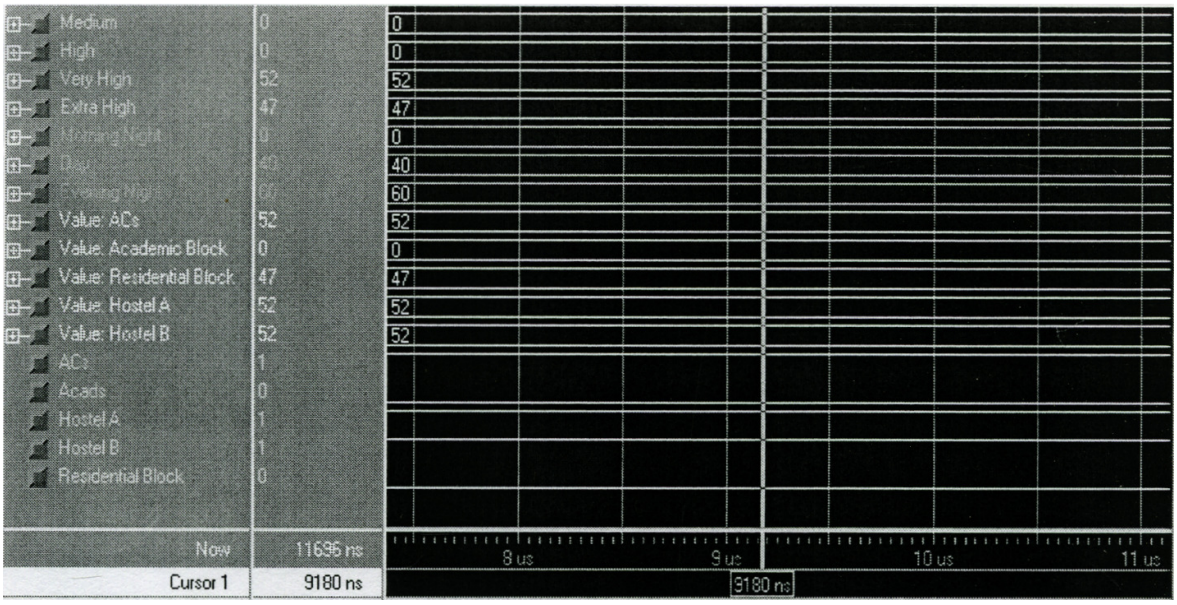


**Fig. 22: Simulation of Rule Matrix**

Time = Evening Night and Load = Very High then off area = Ac, Hostel A and Hostel B

Simulation of overall Structure.

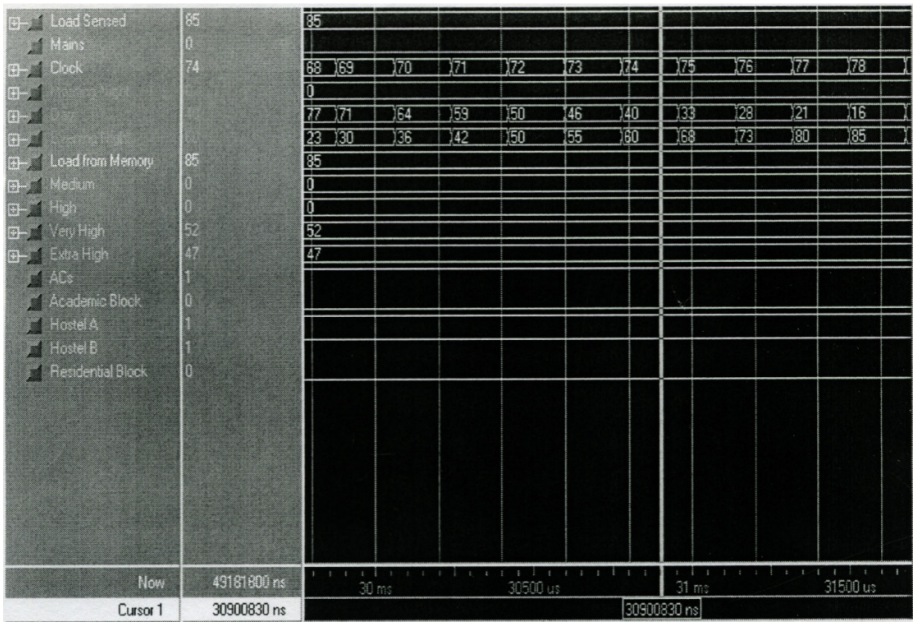


**Fig.: 23 Simulation of overall Structure.**

Fig. Load Sensed = 85 and clock =74 so load = Very High and Time = Evening Night.
Output is Off Area = Acs, Hostel A and Hostel B.

Xilinx Result

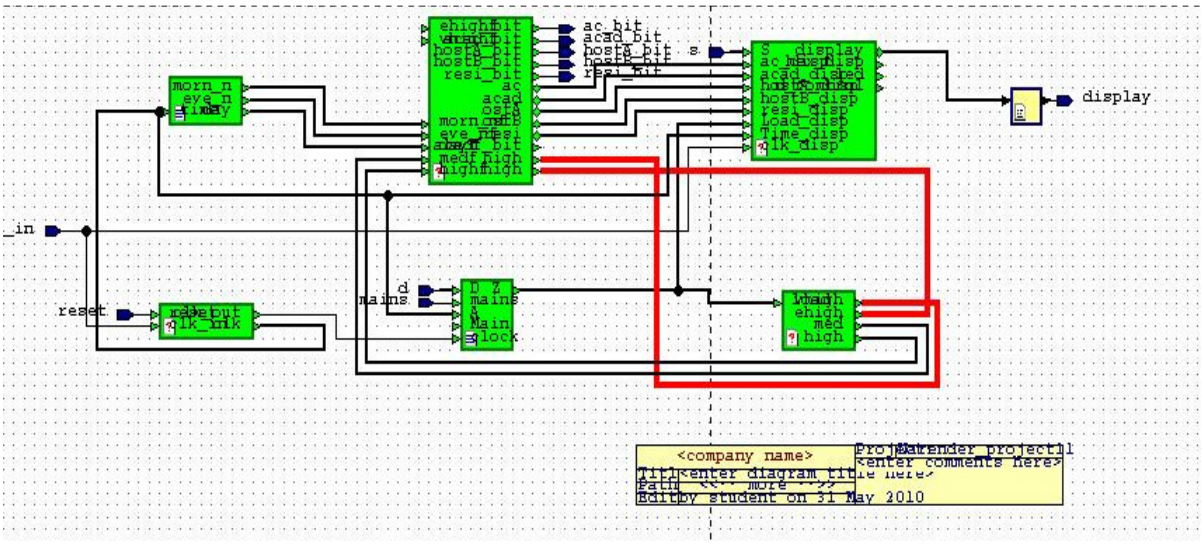


**Fig.24: Overall VHDL Generated Structor of Project.**

# Chapter 5: Conclusion

### 5.1 Device Utilization Summary:

Synthesis and implementation of this FLC was carried out on VIERTEX FPGA Kit using the Xilinx Project Navigator. The synthesis report presents the device utilization summary. The following report is given below:-

……

## Synthesis Option Summary

……

| | |
|---|---|
| Input File Name | : Narenderflc.project |
| Input Format | : Mixed |
| Ignore Synthesis Constrain File | : No |
| **Target Parameters** | |
| Output File Name | : flc |
| Output Format | : NGC |
| Target Device | : xc4vlx25-11ff668 |
| **Source Options** | |
| Top Module Name | : flc |
| Automatic FSM Extraction | : Yes |
| FSM Encoding Algorithm | : Auto |
| FSM Style | : lut |
| RAM Extraction | : Yes |
| RAM Style | : Auto |
| ROM Extraction | : Yes |
| ROM Style | : Auto |
| Mux Extraction | : Yes |
| Mux Style | : Auto |
| Decoder Extraction | : Yes |
| Priority Encoder Extraction | : Yes |
| Shift Registrar Extraction | : Yes |

| | |
|---|---|
| Logical Shifter Extraction | : Yes |
| XOR Collapsing | : Yes |
| Resource Sharing | : Yes |
| Map on DSP48 | : Auto |
| Multiplier Style | : Auto |
| Automatic Register Balancing | : No |

**Target Options**

| | |
|---|---|
| Add IO Buffers | : Yes |
| Global Maximum Fanout | : 500 |
| Add Generic Clock Buffer (BUFG ) | : 32 |
| Register Duplication | : Yes |
| Equivalent register removal | : Yes |
| Slice Packing | : Yes |
| Pack IO Registers into IOBs | : Auto |

**General Options**

| | |
|---|---|
| Optimization Goal | : Speed |
| Optimization Effort | : 1 |
| Keep Hierarchy | : No |
| Global Optimization | : All clock nets |
| RTL Output | : Yes |
| Write Timing Constrains | : No |
| Hierarchy Separator | : _ |
| Bus Delimiter | : < > |
| Case Specifier | : Maintain |
| Slice Utilization Ratio | : 100 |
| Slice Utilization Ratio Delta | : 5 |

**Other Options**

| | |
|---|---|
| Also | :flc.lso |
| Read Cores | : Yes |

Cross_Clock_Analysis : No

Verilog2001 : Yes

Optimize Instantiated Primitives : No

……

## Advanced HDL Synthesis

……

Advance RAM interface……

MAC interface……

Advanced multiplier interface……

Advanced Registered AddSub interface……

DSP optimization……

Dynamic shift registrar interface……

HDL Synthesis Report

Macro Statics

**# ROMs : 5**

16 × 8 –bit ROM : 1

64×8 –bit ROM : 2

128 × 8 –bit ROM : 1

32 × 8 –bit ROM : 1

**# Counters : 1**

30 –bit counter : 1

**# Registers : 102**

10 –bit register : 1

1 –bit register : 1

2 –bit register : 1

14 –bit register : 1

8 –bit register : 98

**# Latches : 3**

14 –bit latch : 1

| | |
|---|---|
| 8 –bit latch | : 2 |
| **# Comparators** | **: 426** |
| 10 –bit comparator greatequal | : 1 |
| 8 –bit comparator equal | : 96 |
| 8 –bit comparator not equal | : 96 |
| 8 –bit comparator greatequal | : 192 |
| 8 –bit comparator less | : 41 |
| **# Multiplexers** | **: 37** |
| 8 –bit 8 – to – 1 multiplexer | : 1 |
| 8 –bit 2 – to – 1 multiplexer | : 36 |
| **# Xors** | **: 1374** |
| 1 –bit Xor 2 | : 1374 |

……

## Final Report

……

Final Results

| | |
|---|---|
| RTL Top Level Output File Name | : narenderflc.ngr |
| Top Level Output File Name | : flc |
| Output Format | : NGC |
| Optimization Goal | : Speed |
| Keep Hierarchy | : No |

Design Statistics

| | |
|---|---|
| **# IOs** | : 30 |

Macro Statistics

| | |
|---|---|
| **ROMs** | : 5 |
| # 128 × 8 –bit ROM | : 1 |
| # 64×8 –bit ROM | : 2 |
| # 32 × 8 –bit ROM | : 1 |
| # 16 × 8 –bit ROM | : 1 |

**Registers** : 102

# 10 –bit register : 1

# 8 –bit register : 98

# 2 –bit register : 1

# 1 –bit register : 1

**Counters** : 1

# 30 –bit counter : 1

**Multiplexers** : 37

# 8 –bit 8 – to – 1 multiplexer : 1

# 8 –bit 2 – to – 1 multiplexer : 36

**Comparators** : 426

# 10 –bit comparator greatequal : 1

# 8 –bit comparator equal : 96

# 8 –bit comparator not equal : 96

#8 –bit comparator greatequal : 192

#8 –bit comparator less : 41

**Cell Usage:**

# BELS : 7030

# GND : 1

# LUT1 : 7

# LUT2 : 154

# LUT_D : 2

# LUT2_L : 144

# LUT3 : 1100

# LUT_D : 831

# LUT_L : 281

# LUT4 : 2639

# LUT4_D : 419

# LUT4_L : 241

| | | |
|---|---|---|
| # MUXCY | : 1086 | |
| # MUUXF5 | : 56 | |
| # MUXF6 | : 31 | |
| # MUXF7 | : 7 | |
| # VCC | : 1 | |
| # XORCY | : 30 | |
| FlipFlops/Latches | : 856 | |
| # FD | : 19 | |
| # FDC | : 10 | |
| # FDCE | : 1 | |
| # FDCPE | : 30 | |
| # FDE | : 776 | |
| # FDR | : 3 | |
| # FDS | : 1 | |
| # LD | : 12 | |
| # LD_1 | : 4 | |
| Clock Buffers | : 22 | |
| # BUFG | : 1 | |
| # BUFGP | : 1 | |
| IO Buffers | : 29 | |
| # IBUF | : 13 | |
| # OBUF | : 16 | |
| Selected Device | : 4vlx25ff668 -11 | |
| Number of Slices | : 3208 out of 10752 | 29% |
| Number of 4 input LUTs | : 856 out of 21504 | 3% |
| Number of bonded IOBs | : 5818 out of 21504 | 27% |
| Number of GCLKs | : 2 out of 32 | 6% |

Total memory usage is 253168 Kilobytes

## 5.2 Future Scope:

Designing of this Fuzzy logic controller, *TIME* input has been fuzzified into Morning Night, Day and Evening Night as per the North Indian climatic conditions for the winters. It may change for the summers, and to accommodate this variation, 'Time Fuzzifier' module may be retuned. The low cost FLC implemented for the management of captive power for the college campus can be easily extended to management of load shedding at the regional power stations with some improvements in the rule base and retuning of fuzzy sets.

In case of management of regional power stations, instead of low power relays / contactors, high power contactors or circuit breakers will have to be considered. Industry, being a major user of captive power, stands as one of the potential future applications of the designed system. A similar rule base may be designed for the management of captive power for any industry, among its various sections.

## 5.2 Scope of System Improvement:

1. **Seasonal Change:** The system is currently not sensitive towards the change in seasons and thus is sensitive towards climatic changes which lead to alterations in the load. To improve this, another input to the system, temperature, may be defined.

2. **Vocations and Holidays:** Currently the system's behavior is indifferent on holidays as well. The power requirement on holidays are completely different from the regular days. This can be mapped by specifying a different LUT for load on these days and providing with a switchover.

3. **Initial Operation:** The system needs to be run for one full day without power getting disconnected so that the load memory gets appropriate values.

4. **Equations in place of LUT:** From programming and implementation on FPGA point of view, LUTs are an inefficient technique. Instead , equations may be used to specify the position of a line. This would consume lesser space in FPGA.

## Appendix -A

**Source Code of Clock Module**

```
-- VHDL Architecture Narender_project11_lib.Clock_10khz.narender
-- Created:
--          by - student.users (localhost.localdomain)
--          at - 09:43:56 05/05/10
-- using Mentor Graphics HDL Designer(TM) 2005.3 (Build 75)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY Clock_10khz IS
  Port ( Reset : in std_logic;
  Clk_in  : std_logic;
  Clk_out : inout bit;
  Clk : inout bit_vector (7 downto 0));
  END ENTITY Clock_10khz;
 ARCHITECTURE narender OF Clock_10khz IS
  function "+" (a,b: bit_vector (9 downto 0))
  Return bit_vector is
  Variable cin,cout:bit:='0';
  variable sum:bit_vector(9 downto 0):= (others=>'0');
  Begin
  for i in 0 to 9 loop
  Cout:=(a(i) xor b(i)) or (b(i)and cin ) or (a(i) and cin);
  Sum(i):=(a(i) xor b(i))xor cin;
  Cin := cout;
  End loop ;
  Return sum ;
  End "+";
  Function add ( a,b: bit_vector (7 downto 0))
  Return bit_vector is
```

```
Variable cout:bit;
Variable cin :bit:= '0';
Variable sum: bit_vector (7 downto 0):="00000000";
Begin
for i in 0 to 7 loop
Cout:=(a(i) xor b(i))or (b(i)and cin ) or ( a(i) and cin );
Sum (i):=(a(i) xor b(i))xor cin;
Cin := cout;
End loop;
Return sum;
End add;
-- signal declaration , one counter and one "bit" registor.

Signal clk_cnt : unsigned (29 downto 0);
Signal clk_bit : bit;
Signal counter : bit_vector (9 downto 0);
 Begin
Gen_clock : process (clk_in , reset ) is
Begin
-- asynchronous RESET forces clock state to 0
If (reset = '0')then
Clk_cnt <= "00000000000000000000000000000000";
Clk_bit <= '0';
Elsif  rising_edge (clk_in ) then
If (Clk_cnt =4999999) then
Clk_cnt <= "00000000000000000000000000000000";
Clk_bit <= not clk_bit;
Else
Clk_cnt <=clk_cnt+1;
End if;
End if;
End process;
 Clk_out <=clk_bit;
Process(clk_out)
```

```
  Begin
  If (clk_out = '1' and clk_out' event) then
  Counter <= counter + "0000000001";
  End if;
  If counter >= "1110000100" then
  Clk <= add (clk,"00000001");
  Counter <= "0000000000";
  If (clk = "01100000") then
  Clk<= "00000000";
  End if ;
  End if;
  End process;
  --concurrent assignment to connect CLK to clk_out , this acts as a wire
  -- to bond the clk_out signal to the clk pin.
END ARCHITECTURE narender;
```

## Appendix -B

**Source Code of Load Memory**

```
-- VHDL Architecture Narender_project11_lib.Load_memory.load_shading
-- Created:
--          by - student.users (localhost.localdomain)
--          at - 10:16:57 05/12/10
-- using Mentor Graphics HDL Designer(TM) 2005.3 (Build 75)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY Load_memory IS
 Port(
A         : In bit_vector (7 downto 0);

D         : In bit_vector (7 downto 0);
Main : in Bit;
Z : out bit_vector (7 downto 0);
clock : in bit
);
END ENTITY Load_memory;
ARCHITECTURE load_shading OF Load_memory IS
Type location is array (0 to 95) of bit_vector(7 downto 0);
Signal memory : location ;
Function "+" (x,y : bit_vector ( 7 downto 0))
Return bit_vector  is
Variable cout : bit ;
Variable cin : bit := '0';
Variable  sum : bit_vector (7 downto 0):= "00000000";
Begin
For I in 0 to 7 loop
Cout:= (x(i) and y(i)) or (y(i) and cin ) or ( x(i) and cin );
```

```
Sum (i):= x(i) xor y(i) xor cin;
Cin := cout;
End loop ;
Return sum ;
End "+";
Begin
Process (Main , clock)
Variable b: bit_vector (7 downto 0);
Begin
-- for memory write
If (clock= '1' and clock'event) then
For j in 0 to 95 loop
If (a=b)then
Z<=memory (j);
End if ;
b:= b + "00000001";
If (b>= "01100000") then
b:= "00000000";
End if ;
End loop ;
end if ;
If (main = '1' )then
For k in 0 to 95 loop
If (a= b) then
Memory (k)<=d;
End if;
b:=b+ "00000001";
If ( b>=  "01100000") then
b:= "00000000";
End if ;
End loop ;
End if;
End process;
END ARCHITECTURE load_shading;
```

## Appendix -C

**Source Code of Load Fuzzifier**

```
-- VHDL Architecture Narender_project11_lib.Load_fuzzy.Load_fuzzyfier
-- Created:
--          by - student.users (localhost.localdomain)
--          at - 04:32:44 05/27/10
-- using Mentor Graphics HDL Designer(TM) 2005.3 (Build 75)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY Load_fuzzy IS
Port( load : in bit_vector( 7 downto 0);
Med,high,vhigh,ehigh:out bit_vector( 7 downto 0));
END ENTITY Load_fuzzy;
ARCHITECTURE Load_fuzzyfier OF Load_fuzzy IS
BEGIN
Process(load)
Begin
  Case load is
                When "00000001" => med <= "00000010"; high<= "00000000";
                        vhigh <="00000000"; ehigh<="00000000";
                When "00000010" => med <= "00000110"; high<= "00000000";
                        Vhigh <= "00000000"; ehigh <= "00000000";
                When "00000011" => med <= "00001010"; high<= "00000000";
                        Vhigh <= "00000000"; ehigh <= "00000000";
                When "00000100" => med <= "00001111"; high<= "00000000";
                        Vhigh <= "00000000"; ehigh <= "00000000";
                When "00000101" => med <= "00010110"; high<= "00000000";
                        Vhigh <= "00000000"; ehigh <= "00000000";
                When "00000110" => med <= "00010110"; high<= "00000000";
                        Vhigh <= "00000000"; ehigh <= "00000000";
```

```
When "00000111" => med <= "00011010"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001000" => med <= "00100000"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001001" => med <= "00100011"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001010" => med <= "00100101"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001011" => med <= "00101010"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001100" => med <= "00101110"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001101" => med <= "00110010"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001110" => med <= "00110111"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00001111" => med <= "00111011"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00010000" => med <= "00111110"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00010001" => med <= "01000011"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00010011" => med <= "01001011"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00010100" => med <= "01001111"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00010101" => med <= "01010011"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00010110" => med <= "01010111"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00010111" => med <= "01011011"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00011000" => med <= "01100000"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
```

```
When "00011001" => med <= "01100100"; high<= "00000010";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00011010" => med <= "01100001"; high<= "00000110";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00011011" => med <= "01011100"; high<= "00001010";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00011100" => med <= "01011000"; high<= "00001111";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00011101" => med <= "01010100"; high<= "00010100";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00011110" => med <= "01010000"; high<= "00010110";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00011111" => med <= "01001100"; high<= "00011010";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100000" => med <= "01001000"; high<= "00011010";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100001" => med <= "01000100"; high<= "00100000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100010" => med <= "01000000"; high<= "00100011";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100011" => med <= "00111100"; high<= "00100111";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100100" => med <= "00111000"; high<= "00101010";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100101" => med <= "00110101"; high<= "00101110";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100110" => med <= "00110010"; high<= "00110010";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00100111" => med <= "00101100"; high<= "00110111";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00101000" => med <= "00101000"; high<= "00111011";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00101001" => med <= "00100100"; high<= "00111110";
        Vhigh <= "00000000"; ehigh <= "00000000";
```

```
When "00101010" => med <= "00100000"; high<= "01000011";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00101011" => med <= "00011101"; high<= "01000111";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00101100" => med <= "00011000"; high<= "01001011";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00101101" => med <= "00010100"; high<= "01001111";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00101110" => med <= "00010000"; high<= "01010011";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00101111" => med <= "00001100"; high<= "01010111";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00110000" => med <= "00001000"; high<= "01011011";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00110001" => med <= "00000100"; high<= "01100000";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00110010" => med <= "00000100"; high<= "01100100";
        Vhigh <= "00000000"; ehigh <= "00000000";
When "00110011" => med <= "00000000"; high<= "01100001";
        Vhigh <= "00000010"; ehigh <= "00000000";
When "00110100" => med <= "00000000"; high<= "01011100";
        Vhigh <= "00000110"; ehigh <= "00000000";
When "00110101" => med <= "00000000"; high<= "01011000";
        Vhigh <= "00001010"; ehigh <= "00000000";
When "00110110" => med <= "00000000"; high<= "01010100";
        Vhigh <= "00001111"; ehigh <= "00000000";
When "00110111" => med <= "00000000"; high<= "01010000";
        Vhigh <= "00010100"; ehigh <= "00000000";
When "00111000" => med <= "00000000"; high<= "01001100";
        Vhigh <= "00010110"; ehigh <= "00000000";
When "00111001" => med <= "00000000"; high<= "01001000";
        Vhigh <= "00011010"; ehigh <= "00000000";
When "00111010" => med <= "00000000"; high<= "01000100";
        Vhigh <= "00100000"; ehigh <= "00000000";
```

```
When "00111011" => med <= "00000000"; high<= "01000000";
        Vhigh <= "00100011"; ehigh <= "00000000";
When "00111100" => med <= "00000000"; high<= "00111100";
        Vhigh <= "00100111"; ehigh <= "00000000";
When "00111101" => med <= "00000000"; high<= "00111000";
        Vhigh <= "00101010"; ehigh <= "00000000";
When "00111110" => med <= "00000000"; high<= "00110101";
        Vhigh <= "00101110"; ehigh <= "00000000";
When "00111111" => med <= "00000000"; high<= "00110010";
        Vhigh <= "00110010"; ehigh <= "00000000";
When "01000000" => med <= "00000000"; high<= "00101100";
        Vhigh <= "00110111"; ehigh <= "00000000";
When "01000001" => med <= "00000000"; high<= "00101000";
        Vhigh <= "00111011"; ehigh <= "00000000";
When "01000010" => med <= "00000000"; high<= "00100100";
        Vhigh <= "00111110"; ehigh <= "00000000";
When "01000011" => med <= "00000000"; high<= "00100000";
        Vhigh <= "01000011"; ehigh <= "00000000";
When "01000100" => med <= "00000000"; high<= "00011101";
        Vhigh <= "01000111"; ehigh <= "00000000";
When "01000101" => med <= "00000000"; high<= "00011000";
        Vhigh <= "01000111"; ehigh <= "00000000";
When "01000110" => med <= "00000000"; high<= "00010100";
        Vhigh <= "01001111"; ehigh <= "00000000";
When "01000111" => med <= "00000000"; high<= "00010000";
        Vhigh <= "01010011"; ehigh <= "00000000";
When "01001000" => med <= "00000000"; high<= "00001100";
        Vhigh <= "01010111"; ehigh <= "00000000";
When "01001001" => med <= "00000000"; high<= "00001000";
        Vhigh <= "01011011"; ehigh <= "00000000";
When "01001010" => med <= "00000000"; high<= "00000100";
        Vhigh <= "01100000"; ehigh <= "00000000";
When "01001011" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01100100"; ehigh <= "00000000";
```

```
When "01001100" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01011111"; ehigh <= "00000100";
When "01001101" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01011010"; ehigh <= "00001010";
When "01001110" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01010101"; ehigh <= "00001110";
When "01001111" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01010000"; ehigh <= "00010011";
When "01010000" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01001100"; ehigh <= "00011000";
When "01010001" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01000111"; ehigh <= "00011101";
When "01010010" => med <= "00000000"; high<= "00000000";
        Vhigh <= "01000010"; ehigh <= "00100010";
When "01010011" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00111101"; ehigh <= "00100111";
When "01010100" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00111001"; ehigh <= "00101011";
When "01010101" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00110100"; ehigh <= "00101111";
When "01010110" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00101111"; ehigh <= "00110100";
When "01010111" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00101010"; ehigh <= "00111001";
When "01011000" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00100110"; ehigh <= "00111110";
When "01011001" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00100000"; ehigh <= "01000010";
When "01011010" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00011100"; ehigh <= "01000110";
When "01011011" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00010111"; ehigh <= "01001011";
When "01011110" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00001001"; ehigh <= "01011010";
```

```
When "01011111" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00000100"; ehigh <= "01011111";
When "01100000" => med <= "00000000"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "01100100";
    When others => med <= "00000000"; high<= "00000000";
        Vhigh <= "00000000"; ehigh <= "00000000";
        end case;
        end process;
        END ARCHITECTURE Load_fuzzyfier;
```

## Appendix -D

**Source Code of Time Fuzzifier**

```
-- VHDL Architecture Narender_project11_lib.time.time_fuzzifier
-- Created:
--          by - student.users (localhost.localdomain)
--          at - 04:02:43 05/24/10
-- using Mentor Graphics HDL Designer(TM) 2005.3 (Build 75)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY time IS
port (time:in bit_vector(7 downto 0);
morn_n,day,eve_n:out bit_vector (7 downto 0));
END ENTITY time;

ARCHITECTURE time_fuzzifier OF time IS
BEGIN
process(time)
begin
case time is
when"00000000"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00000001"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00000010"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00000011"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00000100"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00000101"=>morn_n<="01100100";  day<="00000000";
```

```
eve_n<="00000000";
when"00000110"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00000111"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001000"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001001"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001010"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001011"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001100"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001101"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001110"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00001111"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00010000"=>morn_n<="01100100";  day<="00000000";
eve_n<="00000000";
when"00010001"=>morn_n<="01011110";  day<="00000101";
eve_n<="00000000";
when"00010010"=>morn_n<="01011010";  day<="00001011";
eve_n<="00000000";
when"00010011"=>morn_n<="01010011";  day<="00010010";
eve_n<="00000000";
when"00010100"=>morn_n<="01001011";  day<="00011000";
eve_n<="00000000";
when"00010101"=>morn_n<="01000110";  day<="00011110";
eve_n<="00000000";
when"00010110"=>morn_n<="01000001";  day<="00100011";
```

```
eve_n<="00000000";
when"00010111"=>morn_n<="00111100";  day<="00101010";
eve_n<="00000000";
when"00011000"=>morn_n<="00110010";  day<="00110010";
eve_n<="00000000";
when"00011001"=>morn_n<="00101101";  day<="00110111";
eve_n<="00000000";
when"00011010"=>morn_n<="00101000";  day<="00111100";
eve_n<="00000000";
when"00011011"=>morn_n<="00100011";  day<="01000010";
eve_n<="00000000";
when"00011100"=>morn_n<="00100011";  day<="01000010";
eve_n<="00000000";
when"00011101"=>morn_n<="00010101";  day<="01001111";
eve_n<="00000000";
when"00011110"=>morn_n<="00001111";  day<="01010101";
eve_n<="00000000";
when"00011111"=>morn_n<="00001001";  day<="01011010";
eve_n<="00000000";
when"00100000"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00100001"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00100010"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00100011"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00100100"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00100101"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00100110"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00100111"=>morn_n<="00000000";  day<="01100100";
```

```
eve_n<="00000000";
when"00101000"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00101001"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00101010"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00101011"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00101100"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00101101"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00101110"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00101111"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110000"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110001"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110010"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110011"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110100"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110101"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110110"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00110111"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00111000"=>morn_n<="00000000";  day<="01100100";
```

```
eve_n<="00000000";
when"00111001"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00111010"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00111011"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00111100"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00111101"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00111110"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"00111111"=>morn_n<="00000000";  day<="01100100";
eve_n<="00000000";
when"01000001"=>morn_n<="00000000";  day<="01011111";
eve_n<="00000101";
when"01000010"=>morn_n<="00000000";  day<="01011010";
eve_n<="00001010";
when"01000011"=>morn_n<="00000000";  day<="01010100";
eve_n<="00010001";
when"01000100"=>morn_n<="00000000";  day<="01001101";
eve_n<="00010111";
when"01000101"=>morn_n<="00000000";  day<="01000111";
eve_n<="00011110";
when"01000110"=>morn_n<="00000000";  day<="01000000";
eve_n<="00100100";
when"01000111"=>morn_n<="00000000";  day<="00111011";
eve_n<="00101010";
when"01001000"=>morn_n<="00000000";  day<="00110010";
eve_n<="00110010";
when"01001001"=>morn_n<="00000000";  day<="00101110";
eve_n<="00110111";
when"01001010"=>morn_n<="00000000";  day<="00101000";
```

```
eve_n<="00111100";
when"01001011"=>morn_n<="00000000";  day<="00100001";
eve_n<="01000100";
when"01001100"=>morn_n<="00000000";  day<="00011100";
eve_n<="01001001";
when"01001101"=>morn_n<="00000000";  day<="00010101";
eve_n<="01010000";
when"01001110"=>morn_n<="00000000";  day<="00010000";
eve_n<="01010101";
when"01001111"=>morn_n<="00000000";  day<="00001000";
eve_n<="01011100";
when"01010000"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01010001"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01010010"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01010011"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01010100"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01010101"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01010110"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01010111"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01011000"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01011001"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01011010"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01011011"=>morn_n<="00000000";  day<="00000000";
```

```
eve_n<="01100100";
when"01011100"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01011101"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01011110"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01011111"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when"01100000"=>morn_n<="00000000";  day<="00000000";
eve_n<="01100100";
when others=> morn_n<="00000000";  day<="00000000";
eve_n<="00000000";
end case;
end process;
END ARCHITECTURE time_fuzzifier;
```

## Appendix -E

**Source Code of Rule Matrix**

```
-- VHDL Architecture Narender_project11_lib.rule.ruke_matrix
-- Created:
--          by - student.users (localhost.localdomain)
--          at - 06:38:30 05/25/10
-- using Mentor Graphics HDL Designer(TM) 2005.3 (Build 75)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY rule IS
port (medf,highf,vhighf,morn_nf,daayf,eve_nf: in bit_vector (7 downto 0);
ac,acad,resi,hostA,hostB: inout bit_vector(7 downto 0);
ac_bit:OUT BIT;
acad_bit: OUT BIT;
resi_bit: OUT BIT;
hostA_bit: OUT BIT;
hostB_bit: OUT BIT);
D ENTITY rule;
ARCHITECTURE ruke_matrix OF rule IS
BEGIN
 process(medf,highf,vhighf,morn_nf,dayf,eve_nf)
variable ac1,ac2,ac3,ac4,ac5,ac6,ac7,ac8,ac9,ac10,ac11: bit_vector (7 downto 0);
 variable acad1,acad2,acad3:bit_vector(7 downto 0);
 variable hostA1,hostA2,hostA3,hostA4,hostA5,hostA6: bit_vector(7 downto 0);
 variable hostB1,hostB2,hostB3,hostB4,hostB5: bit_vector(7 downto 0);
 variable temp:bit_vector(7 downto 0);
 begin
 -- minimizing operation for each rule
 if medf<morn_nf then
AC1:=medf;
else
```

```
AC1:=morn_nf;
end if;
if medf<dayf then
hostA1:=medf;
else
hostA1:=dayf;
endif;
if medf<eve_nf then
AC2:medf;
else
AC2:=eve_nf;
end if;
if highf<morn_nf then
AC3:=highf;acad1:=highf;
else
AC3:=morn_nf;acad1:=morn_nf;
end if;
if highf<dayf then
AC4:= highf; hostA2:=highf;
else
AC4:=dayf; hostA2:=dayf;
end if;
if highf<eve_nf then
AC5:=highf; hostB1:=highf;
else ac5:=eve_nf;hostB1:=eve_nf;
end if;
if vhighf<morn_nf then
AC6:=vhighf;Acad2:=vhighf;
else
AC6:=morn_nf;Acad2:=morn_nf;
end if;
if vhighf<dayf then
AC7:=Vhighf; hostA3:=vhighf;host2:=vhighf;
else
```

```
AC7:=dayf;hostA3:=vhighf;hostB2:=dayf;
end if;
if vhighf<eve_nf then
AC8:=vhighf;hostA4:=vhighf;hodtB3:=vhighf;
else
AC8:=eve_nf;hostA4:=eve_nf;hostB3:=eve_nf;
end if;
if enightf<morn_nf then
AC9:=ehighf;Acad3:=ehighf;resi1:=ehighf1;
else
AC9:=morn_nf;Acad3:=morn_nf;resi1:=morn_nf;
end if;
if ehighf<dayf then
AC10:=ehighf;hostA5:=ehighf;hostB4:=ehighf;resi2:=ehighf;
else
AC10:=dayf;hostA5:=dayf;hostB4:=dayf;
end if;
if ehighf<eve_nf then
AC11:=ehighf;hostA6:=ehighf;hostB5:=ehighf;resi3:=ehighf;
else
AC11:=eve_nf;hostA6:=eve_nf;hostB5:=eve_nf;resi3:=eve_nf;
endif;
  -- Maximising and aggregation of the outputs
if ac1<ac2 then
temp:=ac2;
else
temp:=ac1;
end if;
if temp<ac2 then
temp:=ac2;
end if;
if temp<ac3 then
temp:=ac3;
end if;
```

```
if temp<ac4 then
temp:=ac4;
end if;
if temp<ac5 then
temp:=ac5;
end if;
if temp<ac6 then
temp:=ac6;
end if;
if temp<ac7 then
temp:=ac7;
end if;
if temp<ac8 then
temp:=ac8;
end if;
if temp<ac9 then
temp:=ac9;
end if;
if temp<ac10 then
emp:=ac10;
end if;
if temp<ac11 then
temp:=ac11;
end if;
ac<=temp;
if(ac<"00110010")then
ac_bit<='0';
else
ac_bit<='1';
end if;
if acad1<acad2 then
temp:=acad2;
else
temp:=acad1;
```

```
end if;
if temp<acad3 then
temp:=acad3;
end if;
acad<=temp;
if (acad<"00110010")then
acad_bit<='0';
else
acad_bit<='1';
end if;
if resi1<resi2 then
temp:=resi2;
else
temp:=resi1;
endif;
if temp<resi3 then
temp:=resi3;
endif;
resi<=temp;
if(resi<"00110010")then
 resi_bit<='0';
else
resi_bit<='1';
end if;
host_A1<hostA2 then
temp:=hostA2;
else
temp:=hostA1;
end if;
if temp<hostA3 then
temp:=hostA3;
end if;
if temp<hostA4 then
temp:=hostA4;
```

```
end if;
if temp<hostA5 then
temp:=hostA5;
end if;
if temp<hostA6 then
temp:=hostA6;
end if;
hostA<=temp;
if(hostA<"00110010")then
hostA_bit<='0';
else
hostA_bit<='1';
end if;
if hostB1<hostB2 then
temp:=hostB2;
else
temp:=hostB1;
end if;
if temp<hostB3 then
temp:= hostB3;
end if;
if hostB1<hostB4 then
temp:=hostB4;
end if;
if hostB1<hostB5 then
temp:=hostB5;
end if;  hostB<=temp;
if (hostB<"00110010")then
hostB_bit<='0';
else
hostB_bit<='1';
end if;
end process);
 END rule_matrix;
```

**REFERENCES**


[1] Rajamani K., Hambarge, "Islanding and load shedding schemes for power plants", IEEE Trans. on Power Delivery, vol. 14, Issue 3, pp 805-809, 1999.

[2] El-Osery A.I., Baird D., Bruder S. "Transmission power management in ad hoc networks: issues & advantages", Networking, Sensing and Control, IEEE pp 1043-1092, 2005.

[3] Alfredo Sanz, "Analog Implementation of Fuzzy Controller", proceedings of third IEEE International conference on Fuzzy System, pp 279-283, 1994.

[4] L.A. Zadeh, "Fuzzy Set", Information and Control, vol. 8, pp 338-353, 1965.

[5] Jerry M. Mendel, "Fuzzy Logic System for Engineering: A Tutorial", proceedings of IEEE. vol. 83, pp 345-377, 1995.

[6] Jemika Malik and Anand Ojha, "Design of A Vlsi Fpga Intrgrated Circuit", Proceeding of IEEE Region 5 Workshop, pp 12-14, 2005.

[7] Mohammed Y. Hassan and Waleed F. Sharif, "Design of FPGA based PID-like Fuzzy Controller for Industrial Applications", IAENG International Journal of Computer Science, vol. 34, 2007.

[8] Valentina Salapura, Volker Hamann, "Implementing Fuzzy Control Systems Using VHDL and State charts, Procedding of European Design Automation Conference with EURO-VHDL '96, pp 53-58, 1996.

[9] D. Driankov, H. Hellendoorn, and M. Reinfrank, "An Introduction to Fuzzy Control", Narosa Publishing House, New Delhi, 1997.

[10] J Lu, M H Nehrir, Donald A. Pierre, "A Fuzzy Logic Based Adaptive Power System Stabilizer",IEEE power engineering society winter meeting, vol.1, pp 92- 95, 1991.

[11] Shi Jaun, L H Herron and A Kalam, "Comparison of Fuzzy Logic Based and Rule Based Power System Stabilizer", IEEE Conference on control application, pp 692- 697, 1992.

[12] K A El-Metwally, O P Malik, "Fuzzy Logic Based Power System Stabilizer", IEEE Proc- Gener.Transm. Distri., vol. 142, No 3, 1995.

[13] H.Taliyat, J Sadeh and R Ghazi "Design of Augmented Fuzzy Logic Power System Stabilizer to Enhance Power System Stability" IEEE Trans., on energy conversion, vol. 11, No. 1, pp 97-103, 1996.

[14] A L Elshafei and K El-Metwally, “Power system stabilization via adaptive fuzzy logic control”, IEEE Inernational symposium on intelligent control, pp 89-94, 1997.

[15] N Hosseinzadeh and A Kalam, “A Direct Adaptive Fuzzy Power System Stabilizer”, IEEE Transactions On Energy Conversion, vol.14, No.4, pp 1564-1571, 1999.

[16] P Kundur, “Effective use of Power Stabilizers for Enhancement of Power System Reliability”, IEEE Power engineering society Summer meeting, vol. 1, pp 96-103, 1999.

[17] N Hosseinzadeh and A Kalam, “ Rule Based Fuzzy Power System Stabilizer Tuned by a Neural Network”, IEEE transactions on energy conversion, vol.14,No.3, pp 773-779, 1999.

[18] D. Driankov, H. Hellendoorn, and M. Reinfrank.” An Introduction to Fuzzy Control”. Springer-Verlag, New York, 1993

[19] J. Yen, R. Langari, “Fuzzy Logic: Intelligent Control and Information”, Pearson Education (Singapore) Pvt. Ltd., India, 2004.

[20] B. Kosko, “Fuzzy Engineering”, Prentice Hall International Inc., 1999.

[21] Deming Chen, Jason Cong, and Peichan Pan, “FPGA Design Automation: A Survey”, Foundations and Trends in Electronic Design Automation Volume 1 Issue 3, 2006.

[22] FPGA Advantage Bookcase: Product Manual, FPGA Advantage, Version 6.2, Mentor Graphics Inc. (USA).

[23] Peter J. Ashenden, “The designer’s guide to VHDL”, Second edition, Morgan Kauffman, San Francisco, 1996.