YAŞAR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

PHD THESIS

FPGA BASED CONTROL AND RECONFIGURABLE MECHANISM IN WSN

AHMED KHAZAL YOUNIS, AL-AZZAWI

THESIS ADVISOR: ASSOC. PROF. TUNCAY ERCAN

COMPUTER ENGINEERING

PRESENTATION DATE: 03.05.2019

BORNOVA / İZMİR
MAY 2019

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the Doctor of Philosophy.

Jury Members:	**Signature:**
Assoc. Prof. Dr. Tuncay ERCAN Yaşar University	[illegible]
Prof. Dr. Mehmet ÜNLÜTÜRK Yaşar University	[illegible]
Assoc. Prof. Dr. Y. Murat ERTEN Izmir Institute of Technology	[illegible]
Asst. Prof. Dr. Samsun BAŞARICI Adnan Menderes University	[illegible]
Asst. Prof. Dr. Mahir KUTAY Yaşar University	[illegible]

[illegible]

Prof. Dr. Cüneyt GÜZELİŞ
Director of the Graduate School

ABSTRACT

FPGA BASED CONTROL AND RECONFIGURABLE MECHANISM IN WSN

Al-Azzawi, Ahmed Khazal Younis
PHD, Computer Engineering
Advisor: Assoc. Prof. Tuncay ERCAN
May 2019

Wireless Sensor Networks (WSNs) enable various solutions for many application fields. They have acquired large popularity because of their low cost, scalability, easily distributed and maintainable features compared to the traditional infrastructure of monitoring solutions. Generally, any WSN consists of a huge number of distributed sensor nodes that can measure a specific physical phenomenon, perform required tasks and enable monitoring of infrastructure-less deployment. A wireless sensor node is constrained by its cost, energy, and size. In order to overcome these limitations, this thesis aims to introduce a smart design and multi-tasking for wireless sensor networks by extending the capability of re-configurability and enhancing the computational power of the sensor nodes. These facilities can be integrated in a typical WSN and IoT (Internet of Things) environment by incorporating an FPGA (Field Programmable Gate Array) device. Enhancing the computational power of data analysis is done by integrating multi-core of adaptive neuro-fuzzy algorithms in the FPGA device. This thesis contributes into the scientific knowledge with two new modifications in the ANFIS (Adaptive Neuro-fuzzy Inferences System algorithm) by using the momentum factor that will accelerate the training phase and adding a new hidden layer to decrease the number of adjustable linear parameters in the FPGA resources. The neuro-fuzzy approach is a kind of soft computation systems that combine between the learning capability of artificial neural networks and the powerful inference system in fuzzy logic. These two techniques are utilized in order to deal with complex problems when the traditional methods cannot provide a simple and precise solution. Later, the multi-hardware cores of the modified algorithms are comprised within the architecture of FPGA by using the different sensor data sets. HW/SW cores in the FPGA are used to perform different tasks such as classification, prediction, alerts, and decision-making.

On the other hand, the modified ANFIS algorithm will extend the capability of MATLAB toolbox for software developers, and the newly generated IP library in the FPGA that will combine multi-server and multi-client devices will be ready to use by researchers in different IoT implementations. The simulation results of the hardware architecture proposed in this thesis provide significant contributions to the state of the art by providing additional insights and a deeper understanding when compared with existing academic and commercial examples.

Key Words: adaptive systems, neuro-fuzzy system, re-configurability, ANFIS, FPGA, smart systems, embedded systems, smart control.

ÖZ

KABLOSUZ ALGILAYICI AĞLAR İÇİN FPGA TABANLI KONTROL VE YAPILANDIRILABİLİR MEKANİZMA

Al-Azzawi, Ahmed Khazal Younis
Doktora Tezi, Bilgisayar Mühendisliği
Danışman: Doç. Dr. Tuncay ERCAN
Mayıs 2019

Kablosuz Algılayıcı Ağlar (KAA) birçok uygulama alanı için farklı çözümler sağlarlar. Bu ağların geleneksel izleme altyapılarıyla karşılaştırıldıklarında sahip oldukları düşük maliyet, ölçeklenebilirlik, kolaylıkla dağıtılabilir olmaları ve sürdürülebilirlikleri nedeniyle kullanımları da oldukça artmıştır. Genel olarak, KAA'lar kullanıldıkları ortamla ilgili olarak fiziksel değişiklikleri ölçebilecek çok fazla sayıda algılayıcı düğümler içerebilir, önceden belirlenmiş görevleri yerine getirebilir ve altyapısı olmayan yerlerde maliyet, enerji ve işlemci gibi kısıtlarına rağmen izleme yapabilirler. Kablosuz bir algılayıcı düğümün maliyet, enerji ve boyut gibi bazı sınırlamalarını kısmen veya tamamen ortadan kaldırabilmek için, bu tezimizde algılayıcı düğümlerin çoklu görevleri yerine getirebilmeleri için işlemci gücünü artıracak, tekrar yapılandırılabilme kabiliyetini yükseltecek akıllı bir tasarım hedeflenmiştir. Bu da tipik bir KAA veya Nesnelerin İnterneti (IoT-Internet of Things) ortamında FPGA (Alan Programlanabilir Kapı Dizisi) cihazlarının kullanılmasıyla gerçekleştirilebilir. Böyle bir ortamda veri analizi için işlemci gücünün arttırılması FPGA cihazı üzerinde uyarlanabilir yapay sinir ve bulanık algoritmaların birlikte çalıştırılmasıyla gerçekleştirilebilir. Bu tez bilimsel bilgiye ANFIS algoritmalarındaki iki yeni değişiklikle katkıda bulunmaktadır. Birincisi veri setlerine olan tanışıklığın sağlandığı eğitim safhasını hızlandıracak momentum faktörünün eklenmesi, diğeri FPGA üzerindeki kaynakların kullanımını etkinleştirecek olan doğrusal parametre sayısını azaltacak ilave bir gizli katmanın eklenmesidir. Sinir-Bulanık yaklaşımlar yapar sinir ağlarındaki öğrenme ve bulanık sistemlerdeki güçlü çıkarım yapabilme yeteneklerini öne çıkartan yazılım tabanlı teknikleri içerirler. Bu iki farklı teknik, geleneksel yöntemlerin basit ve kesin bir çözüm sağlayamadığı karmaşık problemleri çözebilmeyi mümkün kılmaktadır. Daha sonra değiştirilmiş olan bu algoritmalar farklı veri setlerini kullanarak FPGA mimarisi içinde uygulanır. FPGA içindeki donanım ve yazılım bileşenleri sınıflandırma, kestirme, alarm ve karar verme gibi farklı görevleri yerine getirebilir. Diğer yandan, değiştirilmiş ANFIS algoritması yazılım geliştiriciler için MATLAB araç kutusunun yeteneğini arttıracak, FPGA cihazı üzerinde çoklu-kullanıcı ve çoklu-sunucu ortamları için geliştirilen yeni kütüphane de IoT uygulamalarında araştırmacılar tarafından kullanılmaya hazır olacaktır. Bu tezde önerilen donanım yapısı ile ilgili benzetim sonuçları, akademik ve ticari örneklerle karşılaştırıldığında, mevcut bilgiye ve anlayışa en güncel bilgilerle yeni bir derinlik katıldığını ve önemli bilimsel katkılar sağladığını göstermektedir.

Anahtar Kelimeler: uyarlanabilir sistemler, sinir-bulanık sistem, yeniden yapılandırma, ANFIS, FPGA, akıllı sistemler, gömülü sistemler, akıllı kontrol.

ACKNOWLEDGEMENTS

I would like to express my enduring love to my parents, who are always supportive, loving and caring to me in every possible way in my life.

Foremost, I would like to express my sincere gratitude to my advisor Assoc. Prof. Dr. Tuncay Ercan for the continuous support of my Ph.D. study and related research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me a lot in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Dr. Mehmet Ünlütürk and Assoc. Prof. Dr. Y. Murat Erten for their insightful comments, encouragement, and for their hard questions that incented me to widen my research from various perspectives.

Last but not the least; I would like to thank my family: my mother, my wife, and my three wonderful children.

Ahmed Khazal Younis Al-Azzawi

Izmir, 2019

TEXT OF OATH

I declare and honestly confirm that my study, titled "FPGA BASED CONTROL AND RECONFIGURABLE MECHANISM IN WSN" and presented as a PhD Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Ahmed Khazal Younis Al-Azzawi

Signature

[illegible]

....

May 3, 2019

TABLE OF CONTENTS

LIST OF FIGURES

Figure 4.27. Testing Temperature Data using Modified ANFIS and ANFIS Toolbox 90

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

SSR	Structure of Scientific Revolution
NS	Normal Science
EOS	Extra Ordinary Science
ADC	Analogue-To-Digital Converter
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
APSoC	All-Programmable System-On-Chip
APU	Application Processing Unit
ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BP	Back-Propagation
CLB	Configurable Logic Block
COA	Centroid of Area
COS	Center of Sums technique
DAC	Digital-To-Analogue Converter
DHCP	Dynamic Host Configuration Protocol
DSP	Digital Signal Processing block
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMIO	Extended Multiplexed Input / Output
ESL	Electronic System Level
FF	Flip-Flop
FFT	Fast Fourier Transform
FIFO	First In First Out
FIR	Finite Impulse Response filter

FLS	Fuzzy Logic System
FPGA	Field Programmable Gate Array
GDM	Gradient Descent
GEM	Gigabit Ethernet Controllers
GPIO	General-Purpose Input /Output
GUI	Graphical User Interface
HAS	Home Automation System
HDL	Hardware Description Language
HLS	Vivado High-Level Synthesis
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
ICMP	Internet Control Message Protocol
IDE	Integrated Design Environment
IGMP	Internet Group Message Protocol
IOBs	Input/output blocks
IP	Intellectual Property
IP	Internet Protocol
iPSO	improved Particle Swarm Optimization
IT	Information Technology
LSM	Linear Least Squares Algorithm
LUT	Look-Up Table
lwIP	Light weight IP
MAC	Media Access Control
MF	Membership Function
MIO	Multiplexed Input / Output
MOM	Mean of Maxima Technique

MSE	Mean Squared Error
NFS	Neuro-Fuzzy System
PC	Personal Computer
PL	Programmable Logic
PLD	Programmable Logic Devices
PS	Processing System
RAM	Random Access Memory
RMSE	Root-Mean-Square Error
RNN	Recurrent Neural Network
ROM	Read Only Memory
RTL	Register Transfer Level
SDK	Software Development Kit
SDK	Xilinx Software Development Kit
SIMD	Single Instruction Multiple Data
SoC	System-on-Chip
SRAM	Static Random-Access Memory
STD	Standard Deviation
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VHDL	Very high-speed Hardware Description Language
WSN	Wireless Sensor Network

CHAPTER 1
INTRODUCTION

A wireless sensor network (WSN) is a collection of specialized sensor nodes that are connected with a specific communications protocols for observing and recording physical parameters at different locations and then, forward the collected data to the end user. Temperature, humidity, sound intensity, speed, pollutants, wind direction, light, and pressure are examples of the common monitored conditions. Wireless sensor networks provide precise information according to the requirements of the user. Therefore, they enable a huge number of applications such as industrial control systems, health-care, and target tracking in military fields, weather monitoring systems, and home automation systems.

There are many advantages of wireless sensor networks over conventional sensing systems. Because WSNs are deployed in vast scale, they provide more precise data about monitoring objects. In addition, these network systems are also easier to establish and more scalable when compared with traditional wired systems.

Wireless sensor networks have many unique attributes stated in these references (Sohraby, Minoli, and Znati, 2007; Karl and Willig, 2007; Ilyas and Mahgoub, 2004):

- Size: In real applications, the size of sensor nodes is small to reduce the sensor deployment cost.
- Power: The power of Wireless sensor network nodes has a limited energy constraint. However, most utilized applications follow different operating conditions of sensor nodes to maintain the battery power for a long time.
- The large scale of deployment: Large scale of deployment increases the observation accuracy and reduces the communication distance between nodes. This also improves communication quality and increases energy efficiency. A large number of sensor nodes is also a prime enabler for robust communication in mesh wireless sensor network.

- Mobility: Environmental impacts on sensor nodes such as wind or rain, may change their initial position deployment. On the other hand, they can change their position according to the demand for the system properties.
- Unattended operation: Because a huge number of devices are distributed in a WSN, unattended operation and failure-tolerance of every sensor node becomes an essential necessity.
- Low Cost: Since sensor nodes are distributed in wide-range, the cost of a single sensor node is the influential factor in the overall cost of different applications of WSN's. Furthermore, most applications of the wireless sensor network in civilian fields are named price-enabled applications. That means the markets wait for competitive prices to utilize the application.
- Heterogeneity: In general, wireless sensor networks may be composed of various types of nodes. The computational power of some types of sensor nodes may be more than others; the degree of heterogeneity within a WSN has an impact on the complexity and management method of the overall system.

The energy budget limitations of these platforms have led to constructing a very low power system that can perform complex functions with minimum resources utilization. Not only the technology of wireless sensor networks has attracted much interest, but also the ability to implement it in several fields such as agriculture, military, tracking system, biomedicine, etc. Because WSN's have become a universal subject, most of the engineering can discover an advantage or offer a new contribution.

In general, simple and traditional microcontrollers that are used within sensor nodes are not suitable efficiently to handle complex tasks for data processing. To achieve high-performance computing, multi-processor systems are becoming compulsory. However, these kinds of systems do not suitable for low power consumption platforms such as WSNs. The integration of application-specific hardware accelerators with sensor nodes plays a key role to reduce the overall power consumption.

The energy and required time that is used for data processing tasks when implementing them with specific digital logic can be minimized to several times compared to software solutions. Furthermore, the wide range of WSN applications means that a high level of flexibility can be achieved in the underlying hardware. Consequently, a single infrastructure system can be re-utilized for several purposes.

Modern improvements in the of Programmable Logic Devices (PLD) design Have a significant impact on the rate of energy consumption of these chips, which can be used to reconfigure the digital circuit to implement complex tasks of data processing. So, the process of reducing power consumption (static and dynamic power) by integrating it with high logic intensity makes these kinds of logic devices a preferable choice for flexible hardware acceleration. Reconfigurable hardware appears then as a possible solution to provide an adequate level of computation power for any sensor node while providing additional hardware resources for the implementation of different applications in several domains.

1.1. Research Scope and Objectives

In this thesis, we discuss the possibility of integrating FPGA Device in the architecture of sensor nodes. The employment of re-programmable hardware architecture is discussed at multiple levels of abstractions, start from the selection of technology up to employ it in real-world applications and the modification of suitable software programs. In addition, this thesis covers multiple subjects such as architecture design of embedded systems, reconfigurable technologies, and some of the soft computing technologies such as neural network algorithms, fuzzy control systems, and neuro-fuzzy structures. In general, our thesis answers the following questions:

- **What are the benefits and structures of sensor nodes using reconfigurable hardware acceleration?** To answer this question, we need to identify the current structure of the sensor node and potential improvements. The effect of reconfigurable acceleration on the sensor node will be discussed and three types of sensor nodes depend on reconfigurable hardware acceleration are presented.
- **How to combine reconfigurable devices with the structure of sensor nodes and keep the level of power consumption minimum?** To solve this problem, three techniques are implemented to define appropriate structure where total overhead stays unimportant. This structure will be confirmed by designing a hardware model performing typical functions of a sensor node.

- **What kind of soft computing technology that is suitable for the applications of sensor nodes? What kind of software tools are used to implement it within the sensor node?** In this thesis, different approaches of soft computing techniques are presented with the pros and cons of each approach. On the other hand, the abundance of WSN's applications leads to a variety of programming methods where the components of the software and hardware system can be rapidly used for sensor node. So, several tools are utilized to create the sensor node based on programmable devices. As a result, we will introduce a new generic smart sensor node that can be customized for any kind of data processing algorithms. This integration approach provides high flexibility, which can permit the implementation of a wide range of processing tasks. This flexibility can be obtained by supporting dynamic reprogramming which means the function of the sensor node can be replaced at any time.

1.2. Thesis Outline

This thesis is divided into four principal parts. In Part I, the thesis' background and motivation are described, the theoretical of soft computing (artificial neural networks, fuzzy control system, and neuro-fuzzy system) and field programmable gate array that related with this thesis are introduced in Part II. In Part III, we will discuss the proposed modification of neuro-fuzzy algorithm and variety of hardware connotations and tools that are used to design reconfigurable device on a sensor node. The evaluation and the application of the proposed sensor node is presented in Part V. **Figure 1.1** demonstrates the general organization of the thesis. The sections below, introduce a brief outline of the chapters that comprising these four different parts:

Part I: Chapter 1:

- The background, advantages, and Challenges of wireless sensor networks.
- The Research Scope and Objective.
- Literature review of some related works.

Part II: Chapter 2 and Chapter 3:

- Theoretical of neural networks, fuzzy control system, and neuro-fuzzy system
- Theoretical of Field Programmable Gate Array devices

Part III: Chapter 4:

- Modifications of Adaptive Neuro-fuzzy System

Part III: Chapter 5 and Chapter 6:

- Hardware concepts
- Tools that are used to design of reconfigurable device
- Design of Multi-Core Neuro-Fuzzy System based on FPGA
- System Evaluation and Application

Part V: Chapter 7:

- Sums up our thesis by summarizing the main contributions and suggesting outlines for future work.

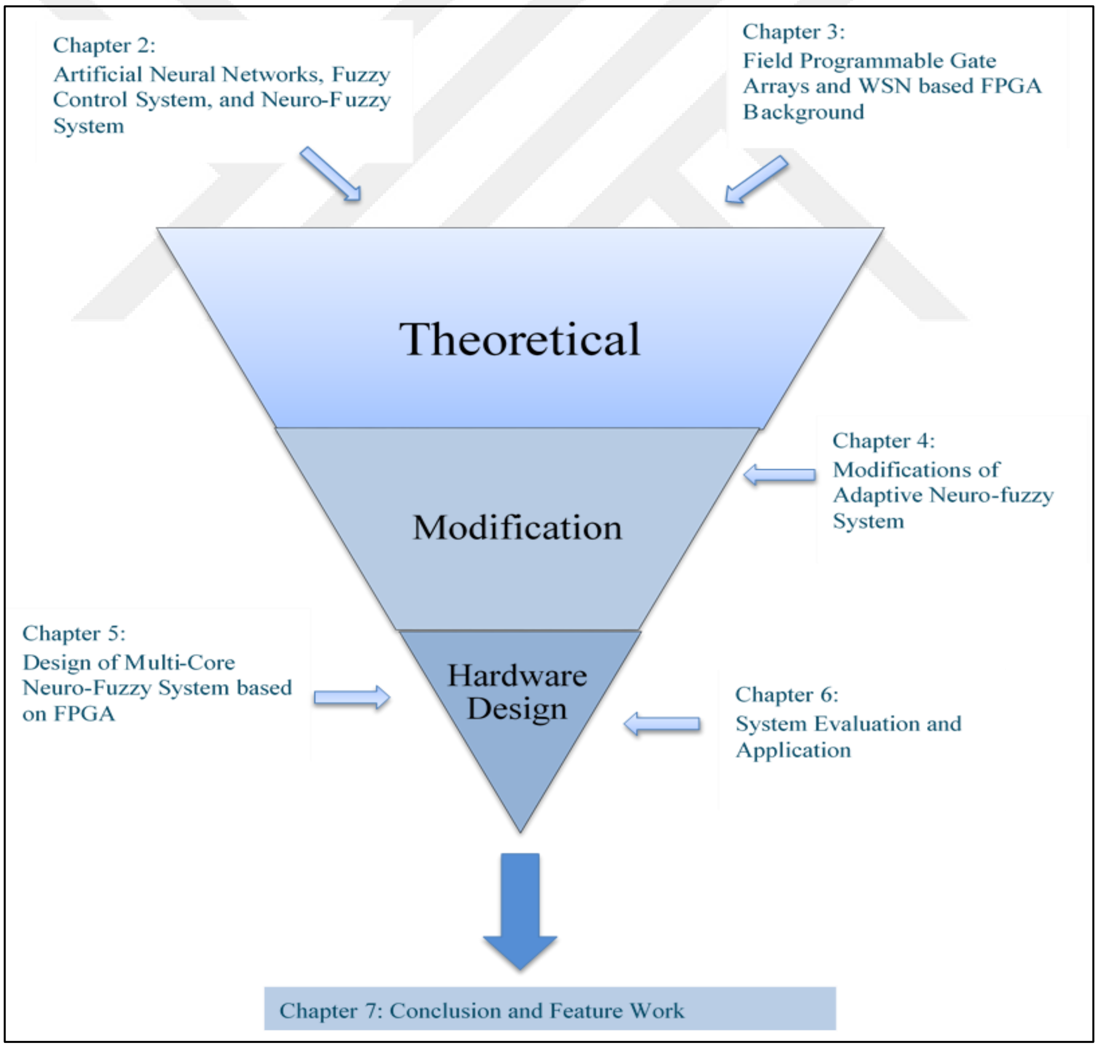

Chapter 2:
Artificial Neural Networks, Fuzzy Control System, and Neuro-Fuzzy System
Chapter 3:
Field Programmable Gate Arrays and WSN based FPGA Background
Theoretical
Modification
Chapter 4:
Modifications of Adaptive Neuro-fuzzy System
Chapter 5:
Design of Multi-Core Neuro-Fuzzy System based on FPGA
Hardware Design
Chapter 6:
System Evaluation and Application
Chapter 7: Conclusion and Feature Work

Figure 1.1. Outline of the Thesis.

1.3. Existing Studies

Field Programmable Gate Arrays include reconfigurable logic elements called Logic Blocks, and smart programmable interconnects that permit these blocks to be wired together. By employing Hardware Description Languages (HDLs) like Verilog or VHDL, designers can configure these reconfigurable blocks to achieve complex logic structures. Several pre-designed FPGA generic component cores exist which permit implementing multipliers, processors, network protocols, etc. So, FPGA devices are extremely flexible especially when they come to designing complex systems.

FPGA technology is being used in several application fields such as telecommunications(Khedkar and Khade, 2017; Yonezawa et al., 2017), signal and image processing(Bhattacharyya, Deprettere, Leupers, and Takala, 2018; Li, Xu, and Zhang, 2017; Sarpotdar, Mathew, Safonova, and Murthy, 2016; Woods, McAllister, Lightbody, and Yi, 2017), automotive applications, robotics(Poudel, Giri, and Munir, 2017; Sahlbach, Thiele, and Ernst, 2017; Velez and Otaegui, 2016), space landing crafts(Guinn et al., 2016; Schrader, Bredemeyer, Mihalachi, Rohde, and Kleine-Ostmann, 2016; Yang et al., 2016).

With the advancements of using reconfigurable FPGA and soft computing techniques, many studies have been done in this field. Cihan, Fuat, and Mehmet in (Karakuzu, Karakaya, and Çavuşlu, 2016) show the ability of implement the meta-heuristic learning algorithms of the neuro-fuzzy system (NFS) on the FPGA based on improved particle swarm optimization (iPSO). According to the authors, this implementation does not need any memory and multiplier usage. Gaussian MF is used as the main membership function. This kind of function has an exponential operation and cannot be realized directly within FPGA. So, the authors proposed a kind of function approximation in order to reduce the number of required hardware resources. The proposed approximation function requires one multiplier, one divider, and three adder modules. Results indicate that proposed implementation and membership function approximation is more effective than other approaches and requires less FPGA hardware resources.

Another parallel implementation of digital ANFIS algorithm using FPGA Zynq processor with single and dual based on VHDL (Very high-speed Hardware Description Language) language was presented in (Darvill, Tisan, and Cirstea, 2017).

Using HDL approach provide fast prototyping and allowing powerful synthesis utilization. Comparison between parallel and serial methodology was done in this article. The authors show that there are many advantages of using parallel architecture over existing serial methodology such as clock cycles and logic utilization in addition to reducing the number of redundant calculation cycles that provide higher scalability compared with the serial approach. Another parallel hardware implementation of ANFIS algorithm was done in (Nadu & Nadu, 2017).

Implementing ANFIS algorithms for real-time control applications is the main challenge. The difficulties come from real-time linear and nonlinear parameters and the adaptation of the control systems (Tămas and Brassai, 2015). Distributing the functions of ANFIS algorithm between multiple dedicated hardware modules work together in parallel can solve these challenges. In other words, the ANFIS IP (Intellectual Property) controller is used to run the main function of the algorithm and the ARM hard-processor in Zynx FPGA kit is used to adapt and train all ANFIS parameters.

Other studies use dual-core hard-processor and a programmable logic part based on Vivado High-Level Synthesis tool with high-level language like C/C++. Parallel ANFIS architecture uses a very high processing speed and the parameter adaptation works in parallel with the output processing in real-time can be found in (Tămas and Brassai, 2015).

The authors in (Łapa, Zalasiński, and Cpałka, 2013) present a new methodology based on an evolutionary strategy for nonlinear modeling in order to perform the reduction of complexity in neuro-fuzzy systems and adapt the parameters and the selection of simple rules for the system structure. This work enhances the operation of ANFIS algorithm and reduces computational tasks and its complexity. There are some other works tried to reduce the data dimensions and the complexity of the neuro-fuzzy systems by combining them with other soft computational methods such as particle swarm optimization (Ghasemi, Kalhori, & Bagherpour, 2016; Rini, Shamsuddin, and Yuhaniz, 2016) or with data mining algorithms like C-means (Çavdar, 2016; Kaur, Dhar, and Guha, 2016).

Authors in (C. Chen, John, Twycross, and Garibaldi, 2016) extend the traditional five layers architecture of the ANFIS algorithm to a 6-layer architecture that allows the fuzzification and the node at the inputs layer to be more explicit. The first layer in the extended architecture has been divided into two layers (layer 0 and 1). The first layer (layer 0) is a fuzzification layer, which is used to generate membership grades of all crisp inputs while the layer 1 calculates the largest membership grade for layer 0 nodes.

According to the existing and reachable literature, we decided that the overall thesis design can be covered up into two folds. The first fold deals with improving the performance of ANFIS algorithm by combining it with other soft algorithms or trying to modify and extend its architecture. The other fold attempts to implement the ANFIS in reconfigurable FPGA based on different methodologies. Our contribution aims to integrate these two folds by extending the architecture of the ANFIS algorithm and using the FPGA to implement the algorithms for real time systems.

On the other hand, several studies use the FPGA for real-time smart home automation systems. In the study of Mohd and Mamun in (Marufuzzaman, Bin, Reaz, & Islam, 2014). authors propose a methodology based smart algorithm for recognizing and analyzing activities of daily life based on FPGA. The modified algorithm aims to classify the home events based on intelligent multi-agent algorithms that interact with each other. The system consists of four prediction agents (event, time, location, power) and one agent for decision-making. Based on the inferential processes within the prediction agents, they forward the necessary commands to the Decision-Making Agent.

Other FPGA prototype of a home automation system based on Artificial Intelligent can be found in (M.B.I. Reaz, Assim, Choong, Hussain, & Mohd-Yasin, 2006). The authors design a system that includes three main units: the prediction unit, the decision unit, and the communication unit. Active-Lezi algorithm for the online predictor is used in the prediction unit to predict possible future actions. While the Q-Learning algorithm that is a kind of reinforcement learning technique is used within the decision unit in order to make the final system decision. Other similar FPGA implementation based artificial intelligent for home automation can be found in (Mamun Bin Ibne Reaz, 2013).

The authors in (Sang-hyun, Lee, and Kyung-il, 2013) propose the methodology that integrates ANFIS with the smart home system to eliminate the difficulties that face the sensors in traditional home systems and generate the intelligent and adaptive system.

The authors design multiple ANFIS algorithms in order to take multiple dynamic decisions. Multiple ANFIS have the ability to deal with the nonlinear and complex events system; as a result, the high efficiency can be achieved in the Home Automation System (HAS).

Another implementation based on FPGA, embedded systems, and web technologies is proposed in (Rusu and Duka, 2017). The proposed system has the ability to control and manage multiple house systems through the internet using a single internet page. According to the authors, the system has many advantages over the similar FPGA implementations like the ability to monitor and control more than one home system through a single web page and LCD panel at the same time. The system has also the flexibility to support different types of communication protocols. Finally, the hardware system is very fast because of the response time between the FPGA kit and the web page is about 500 milliseconds.

To the best of our knowledge, there are some other studies using different wireless communication technologies for FPGA connectivity and HAS. Examples are in the references (Suresh and Mastani, 2018; P Waghmare, Chaure, Chandgude, and Chaudhari, 2017) for Bluetooth communication and (Asadullah and Raza, 2016; Gaikwad, Gabhane, and Golait, 2015; Panigrahi, Qureshi, Saxena, and Reddy, 2016; Sharma et al., 2015) for Wi-Fi in addition to previously used wired technologies.

CHAPTER 2
FIELD PROGRAMMABLE GATE ARRAYS

2.1. Introduction

FPGAs are devices in which almost any digital circuit can be implemented just by using programming languages such as Hardware Description Language or by Schematic Capture. They are used in electronic fields to build complex sequential or combinational logic circuits in order to reduce space and to avoid Application specific ICs design cost. In this chapter, we will give an introduction about the internal structure of this device and how can be used to build a new generation of WSNs.

2.2. Field Programmable Gate Array Structure

Field Programmable Gate Array (FPGA) is a device which consists of a huge number of reconfigurable logic circuits and gives particular reprogrammable circuit. When an FPGA is programmed, the internal logic circuits are connected in a way that allows producing a specific hardware achievement of a software application. In general, FPGA devices do not have any operating systems and they utilize dedicated hardware resources for a specific processing application. Because of the parallel nature of FPGA devices, they do not need to compute the same hardware resources for multiple software applications. So, the performances of fabric FPGA parts of the different applications are not affected in case of much additional processing are added. In addition to a single FPGA device can run multiple control loops at multiple rates and this is one of the main distinctions between the FPGA devices and general processors.

A single FPGA has millions of digital logic circuits and can be used instead of thousands of analog components by incorporating in one integrated circuit (IC). The flexibility of the general structure of the reconfigurable FPGA is explained in Figure 2.1. The FPGA comprises three main reconfigurable parts: Configurable Logic Block (CLB), Programmable Interconnects, and Input/output blocks (IOBs). All these parts can be reconfigured to implement complex applications. reconfigurable IOBs represent as a ring around the FPGA microchip and provide access to the I/O FPGA package pins. While the CLBs distributed as a rectangular behind the IOB(D. Chen, Cong, & Pan, 2006)(Lee & Seshia, 2011).

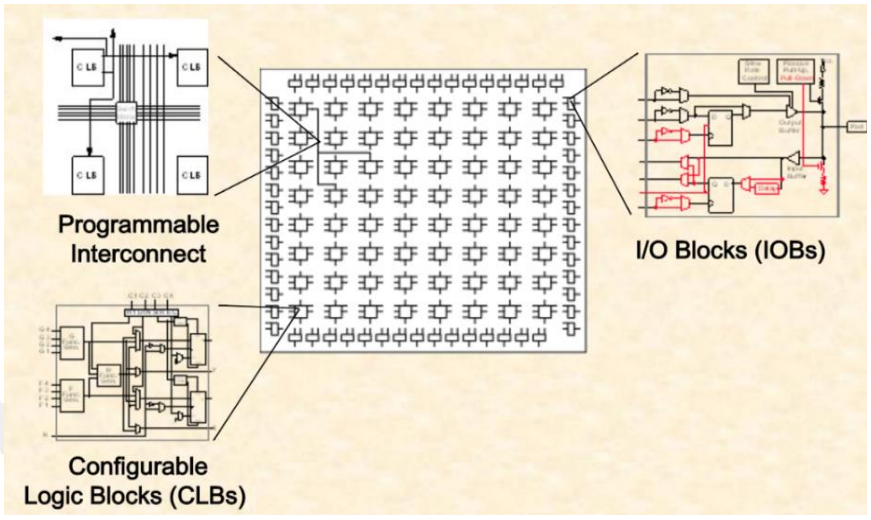


Figure 2.1. General Structure of Reconfigurable FPGA

The Configurable Logic Block of an FPGA can be programmed to implement various kind of combinational functions, sequential functions, or combinations of both of them in a simple way. CLB can be configured by one of these methods(Bobda, 2008):

1. Combinational logic circuits such as AND, OR, XOR, NAND, etc.
2. Lookup tables
3. Multiplexers and De-multiplexers
4. And-OR structure.
5. Transistor pairs

Routing process in FPGAs between CLB is done by programmable interconnects that consists of a various length of wire that can perform the interconnection through configurable switches. The density of CLB that is used for specific application within FPGA is based on length and number of programmable wires applied for the routing process. Simplified routing in internal FPGA architecture is shown in Figure 2.2.

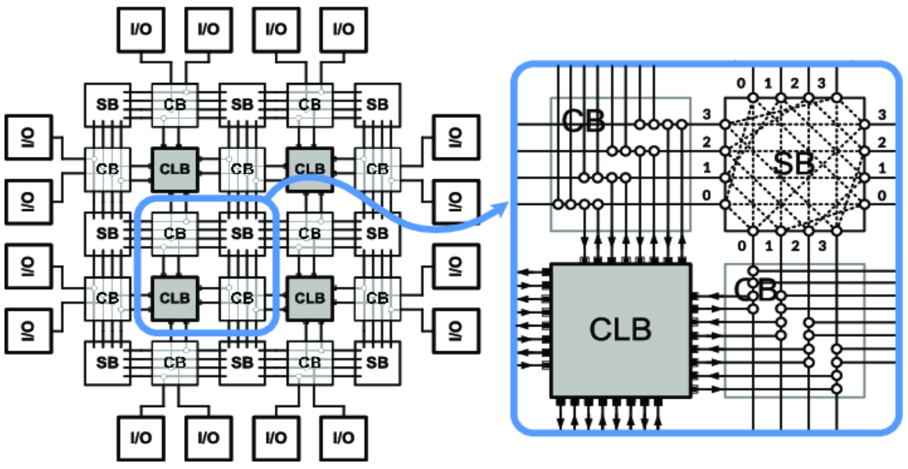


Figure 2.2. Simplified Routing in FPGA Architecture

2.3. FPGA Programming Technologies

There are many technologies of programming FPGA devices that have been used to configure internal architectures. Each technique has different characteristics which have a major effect on the reconfigurable architecture. Some of the techniques comprise SRAM (Static Random-Access Memory), EEPROM (Electrically Erasable Programmable Read-Only Memory), and Anti-fuse(Maxfield, 2004).

2.3.1. FPGA Based SRAM

Static memory cells are the essential elements that are used for FPGAs. Most commercial vendors use static memory (SRAM) based programming technology in their devices. SRAM cells in general, are applied to configure the internal routing of FPGAs which are normally drive by digital multiplexors and to re-configure the CLBs within FPGA devices which are used to perform the various logic functions. Figure 2.3 is an example of a static memory cell that is used in re-configurable devices. In this picture, SRAM stores a bit of data on four transistors using two cross-coupled inverters. The two stable states characterize 0 and 1. During read and write operations another two access transistors are used to manage the availability to a memory cell.

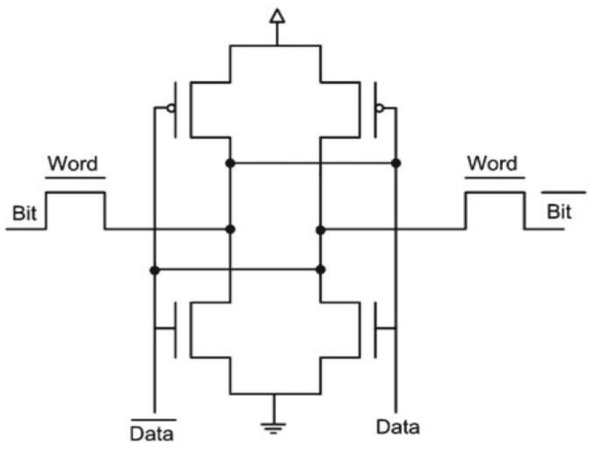


Figure 2.3. Static Memory Cell

Because SRAM-based technology uses the CMOS technology, it becomes a prevalent approach that is used in FPGAs. Also, it improves integration, speed and decreases the power. Using the SRAM technology-based FPGA can be infinitely re-configurable and soldered into the hardware board. The implemented function can be changed very quickly in the field by modifying the information that is stored in the PROM or upload a new application code. But this process takes a huge number of digital look-up table (LUT). Also, the FPGA-based this technique needs to re-program the device every time when the power is turned on.

2.3.2. FPGA Based Anti-fuse

Another alternative to Static memory cells is anti-fuse technology. A high-density interconnect is one of the most important characteristics of this programming technology. This programming technology has a huge number of internal interconnections and that makes the logic application small and efficient. So, the placing and the routing process is much easier and doesn’t take a long time. Also, the resistance and capacitance are small when compared with other programming technologies.

But these kinds of devices are one-time programmable, and we cannot make any change in its design. Another disadvantage of this technology is the demand to integrate all cells of the anti-fuse structure within the integrated circuit (IC). Figure 2.4 demonstrates an example of an array of AND/OR gates connection with two inputs and one output based on anti-fuse technology.

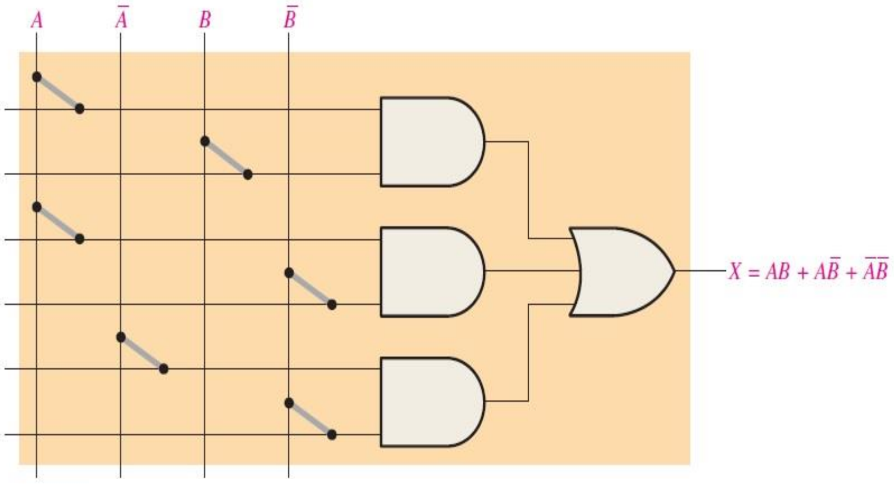


Figure 2.4. AND/OR Structure Based on Antifuse

2.3.3. EEPROM Based

One of the most commonly used and more efficient technologies is the |FPGA based EEPROM cells. This type can be used for two purposes, as s control device like static memory cell or it can be used as a very efficient re-configurable direct switch.

These devices are non-volatile when compared with the SRAM, so they don’t lose its content when power off and do not need any additional PROM for booting. But the process of the EEPROM is more complex and has finite number of re-configurable times when compared with SRAM technology.

2.4. Advantages of using an FPGA

There are many advantages of utilizing FPGA devices over using Application-Specific Integrated Circuit (ASIC) like microcontrollers. The FPGAs are very flexible, reusable, and faster to obtain. Usually, FPGAs are more expensive and have a lower value of production than an ASIC or microprocessor. On the other hand, an FPGA is very cost efficient because of the ability to reprogramming unlimited time for different application tasks. The performance of FPGA is very high because of the parallelized technique that uses in internal architecture. Also, we can build many soft or hard microcontroller inside one FPGA package. Another facility in FPGA is that the ability to re-configurable a portion of its internal fabric while other portions are still working. Any new modification future in the final design can be modified by generating and downloading a new bit-stream file. In the FPGA environment, the user doesn’t need to know every hardware details. The software will carry out everything such as routing and placement. Also, the timing constraint will be done automatically to fit the design specifications. After the design is completed and downloaded it as a bit-stream file, the FPGA will convert to a kind of ASIC (D. Chen et al., 2006; D. Chen et al., 2006 ;D. Chen et al., 2006).

2.5. FPGAs Employed

The Xilinx and Altera are the most common manufacturers in the market which provide the FPGA devices. Other several manufacturers can also provide FPGAs such as Semiconductor, Atmel, etc.

2.5.1. Xilinx:

Xilinx is an American company and it is one of the first and important field-programmable gate array (FPGA) providers. The Virtex and Spartan series are the two families that supplied by this company and they are widely utilized in the market. There are many differences between those two families related to cost and performance.

2.5.1.1. The Spartan series of FPGAs

The Spartan family is designed primarily for the low cost and simple solutions that don’t need a high performance such as the routing algorithms that are used in the wireless sensor networks. Below some of the famous Xilinx Spartan series (Capability, 2016; June and Ram, 2013; Xilinx, 2011):

- The Spartan-6 family is mostly used by sensor designers for wireless communications or for the automotive applications as a lower cost solution and minimum energy consumption.

- The Spartan®-3AN family is a nonvolatile technology that combines between the attributes of previous Spartan-3A family and system-based flash memory technology for programming and data storage.

- The Spartan-3 family is normally utilized to perform real-time applications combined with fuzzy system controller and other applications that need large number of I/O ports.

2.5.1.2. The Virtex series of FPGAs

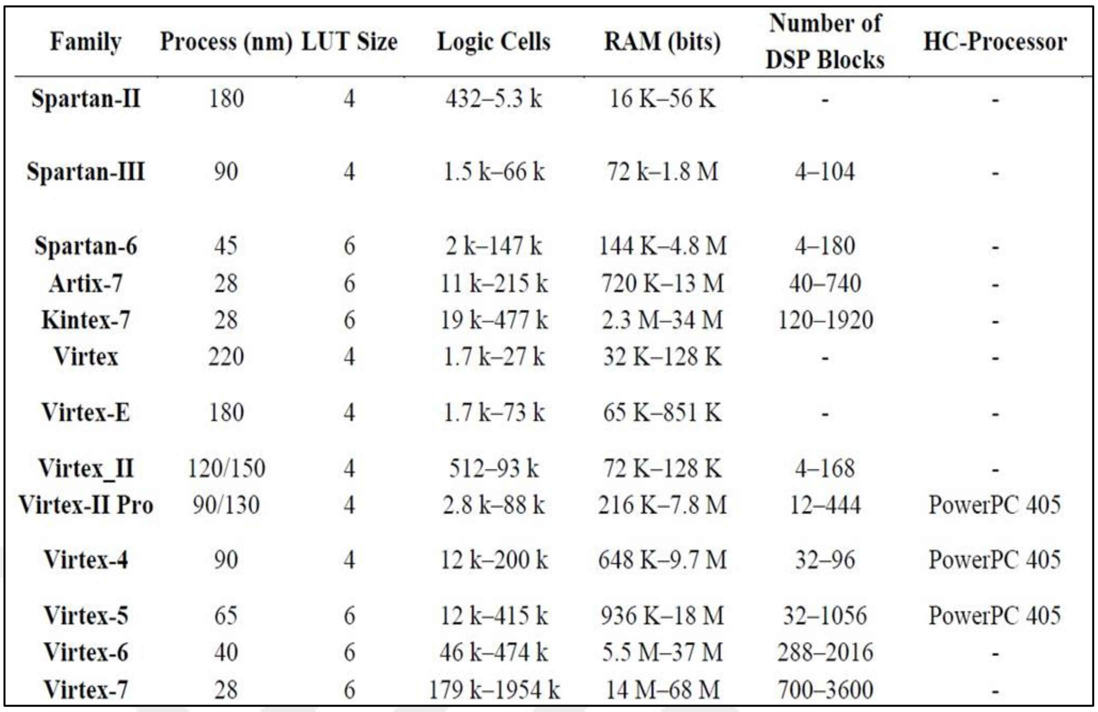
The Virtex series are utilized as a solution for the highest system performance (Xilinx Inc., 2013). Many features are integrated with this series compared with the Spartan series which include Ethernet MAC blocks (Media Access Control), Digital Signal Processing block (DSP), FIFO logic, high-speed transceivers, and PCI-Express controllers. In addition, the Virtex series have many fixed hardware functions that embedded internally and commonly used such as memories, hard cores of microprocessors, serial transceivers, etc.

- The Virtex 7 series is intended to achieve a high-performance at 50% lower power when compared to the Virtex-6 generation devices. Also, memory bandwidth in Virtex-7 is doubled compared to previous Virtex FPGAs generation(Xilinx Inc., 2013).

- The designers of sensor systems try to use the Virtex devices like Virtex-7 or previous generation such as Virtex-II, Virtex-II Pro, etc. in order to implement the complex functions within the sensing systems specifically in wireless sensor networks (WSN) because the FPGA devices are a powerful and robust and a high-performance devices at an acceptable cost.

Table 2.1 shows the characteristics of the different FPGA series of Xilinx.

Table 2.1. Characteristics of Different FPGA Series of Xilinx.



Family	Process (nm)	LUT Size	Logic Cells	RAM (bits)	Number of DSP Blocks	HC-Processor
Spartan-II	180	4	432–5.3 k	16 K–56 K	-	-
Spartan-III	90	4	1.5 k–66 k	72 k–1.8 M	4–104	-
Spartan-6	45	6	2 k–147 k	144 K–4.8 M	4–180	-
Artix-7	28	6	11 k–215 k	720 K–13 M	40–740	-
Kintex-7	28	6	19 k–477 k	2.3 M–34 M	120–1920	-
Virtex	220	4	1.7 k–27 k	32 K–128 K	-	-
Virtex-E	180	4	1.7 k–73 k	65 K–851 K	-	-
Virtex_II	120/150	4	512–93 k	72 K–128 K	4–168	-
Virtex-II Pro	90/130	4	2.8 k–88 k	216 K–7.8 M	12–444	PowerPC 405
Virtex-4	90	4	12 k–200 k	648 K–9.7 M	32–96	PowerPC 405
Virtex-5	65	6	12 k–415 k	936 K–18 M	32–1056	PowerPC 405
Virtex-6	40	6	46 k–474 k	5.5 M–37 M	288–2016	-
Virtex-7	28	6	179 k–1954 k	14 M–68 M	700–3600	-

2.5.2. Altera

Altera is also an American manufacturer that provides programmable logic devices (PLDs) and different types of reconfigurable circuits. Three main families of FPGA devices are provided by Altera: the low-cost Cyclone series, the mid-range Arria series and the last one is the Stratix series that is used for applications that need high-performance. The interesting thing is that all these families are focusing on its soft-core processors that is called NIOS processor series within its FPGA logic devices instead of hard-core processors compared with Xilinx families.

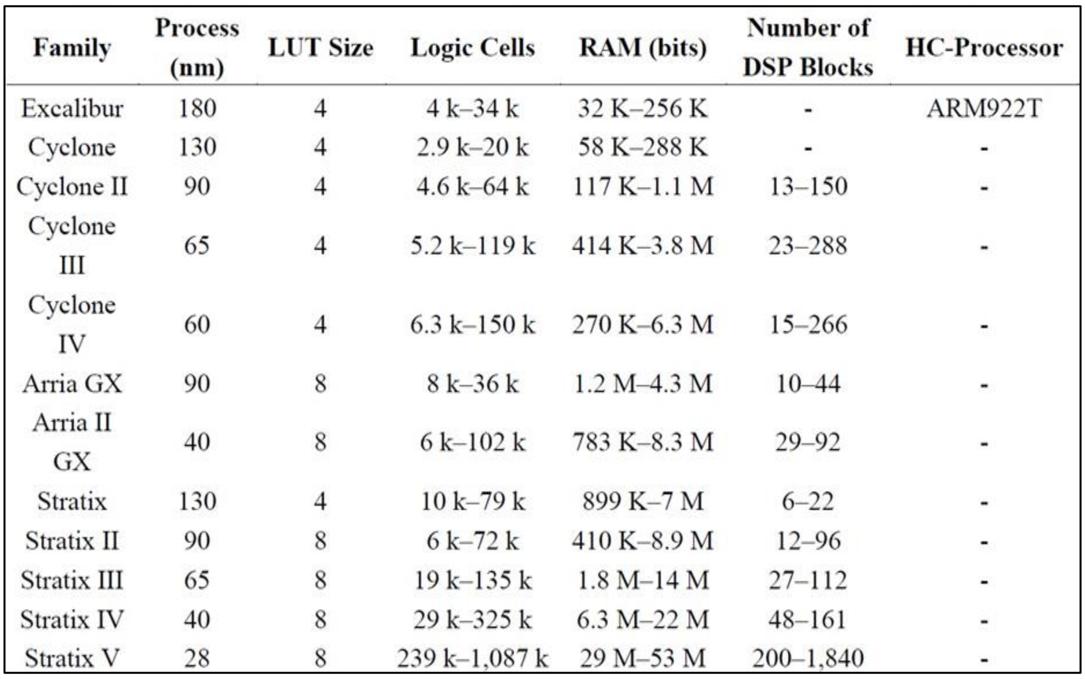
- The Cyclone series: This series was designed for applications that required low-performance and low-cost. It is well suited for many sensor applications such as acquisition real data, hardware image processing, and IR Sensors based Signal Conditioning Circuit. The FPGA Cyclone VI is the recent series from Altera that has 4-input LUT in addition to a register which is associated on the output (Cyclone and Partners, 2003).

- In Stratix FPGAs: They have a similar architecture that is found in the Cyclone series with additional improvements. The LUT in this series has 8 inputs, with many DSP blocks and Ram blocks which can be used as FIFO dual-port RAM or shift registers(Corporation, 2004)(Altera, 2010).

- The Arria series: This series also based on 8-input LUT like the Stratix series. The series is normally utilized for serial communication applications that need a high-performance because it has very high-speed transceiver hardware blocks which are embedded on its internal design. The other characteristics of the Arria FPGA devices are similar to the Stratix devices(Corporation, 2012).

Table 2:2 summarizes characteristics of different Altera series.

Table 2.2. Characteristics of Different Altera Series.



Family	Process (nm)	LUT Size	Logic Cells	RAM (bits)	Number of DSP Blocks	HC-Processor
Excalibur	180	4	4 k–34 k	32 K–256 K	-	ARM922T
Cyclone	130	4	2.9 k–20 k	58 K–288 K	-	-
Cyclone II	90	4	4.6 k–64 k	117 K–1.1 M	13–150	-
Cyclone III	65	4	5.2 k–119 k	414 K–3.8 M	23–288	-
Cyclone IV	60	4	6.3 k–150 k	270 K–6.3 M	15–266	-
Arria GX	90	8	8 k–36 k	1.2 M–4.3 M	10–44	-
Arria II GX	40	8	6 k–102 k	783 K–8.3 M	29–92	-
Stratix	130	4	10 k–79 k	899 K–7 M	6–22	-
Stratix II	90	8	6 k–72 k	410 K–8.9 M	12–96	-
Stratix III	65	8	19 k–135 k	1.8 M–14 M	27–112	-
Stratix IV	40	8	29 k–325 k	6.3 M–22 M	48–161	-
Stratix V	28	8	239 k–1,087 k	29 M–53 M	200–1,840	-

2.6. Xilinx Zinq

Xilinx Zynq is a new generation of FPGA System-on-Chip (SoC). The Zynq devices are primarily manufactured to be used for variety kind of applications because its high flexibility. The device has an embedded dual-core ARM Cortex-A9 processor which build in the internal fabric of FPGA logic.

The dual-core ARM processor in Zynq devices has the capability to run full operating systems whereas the other programmable logics are based on 7-series of the Xilinx FPGA internal architecture (Capability, 2016). The AXI interfaces (Advanced eXtensible Interface) are used to build the Zynq architecture, that supply low-latency connections and high bandwidth between the main elements of the device(Reference, Axi, & Guide, 2014). In other word, the Processing System (PS) part and the Programmable Logic (PL) part can perform the related functions individually without any conflict or overhead between two separate parts. That means, simplifying the system architecture leads to reductions in size and overall cost.

2.6.1. System-on-Chip with Zynq

The concept behind the System-on-Chip is that: a single Integrated Circuit (IC) chip can be utilized to perform all functionality of the hardware system, compared with traditional circuits that need several chips for each function. The term of the SoC normally can be represented as an Application Specific Integrated Circuit (ASIC), that could comprise from analog circuits, digital logic, and other communication components such as radio frequency elements in addition to the digital-to-analog (DACs) and analog-to-digital converters (ADCs) embedded in a signal Integrated Circuit.

From the digital aspect of SoC, the SoC can include all aspects related to a digital system such as memories, microprocessors, interface circuits, high-speed digital elements, and so on. Otherwise, we might need separate devices to realize each function. The SoC technique provides:

- lower cost,
- low power consumption,
- high-level of design security,
- fast-rate of data transfers among the system components,
- small size, and
- high reliability.

A simple graphical diagram of the System-on-Chip is demonstrated in Figure 2.5 (Xilinx, 2018).

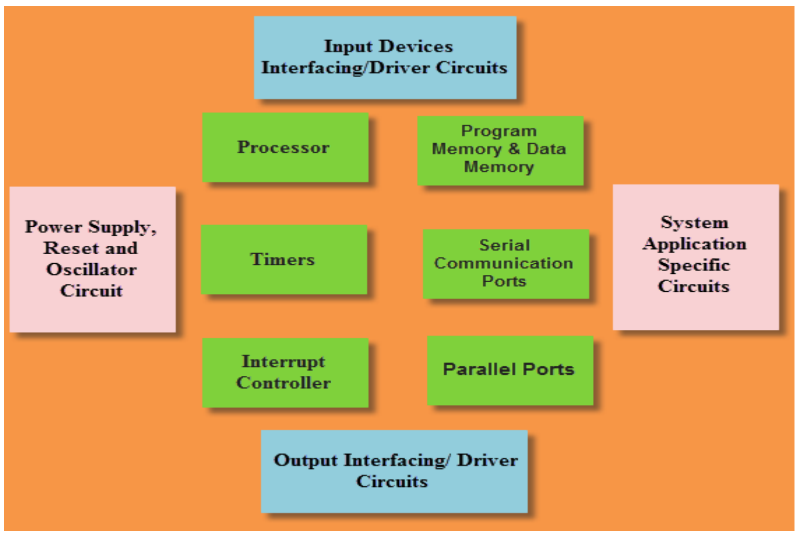

Input Devices Interfacing/Driver Circuits
Processor
Program Memory & Data Memory
Power Supply, Reset and Oscillator Circuit
Timers
Serial Communication Ports
System Application Specific Circuits
Interrupt Controller
Parallel Ports
Output Interfacing/ Driver Circuits

Figure 2.5. Diagram of the System-on-Chip

On the other hand, the ASIC-based SoCs are suffering from disadvantages such as:

(i) The flexibility of the system is low.

(ii) The development of the SoCs system is very high speed and time consumption.

The high cost of system development making this kind of non-recurring engineering effort (and cost) of developing an ASIC are significant, making this type of hardware system suitable only for the application and systems that don't need any future upgrades. Mobile phones, tablets, and PCs are an example of the devices that comprise processors based of ASIC; These kinds of processors normally comprise single or multiple processor cores, interfacing circuits, storage units, and other associated elements (John and Smith, 1997).

All these limitations of ASIC-based SoCs make them incompatible for many significant applications that need the ability of system upgrade and a high level of flexibility. Is it clear the important of the flexibility in our life applications so, motivates the hardware designers to move towards re-programmable devices based SoCs in order to add more flexibility to add the ability of upgrades the functionality of the system and saving the time and cost. Field-Programmable Gate Array (FPGA) is the natural solution and one of the most common examples of re-programmable devices.

FPGAs are adaptable devices which can be programmed to actualize any subjective system. Compared with ASICs devices, the FPGAs offering high-level performance and more flexible hardware platform that can be configured to achieve any desired digital system. There is no risk and very easy when system upgrading is necessary to the latest generation of Xilinx's all-programmable System-on-Chip (SoC).

Now, the FPGA-based Zynq device gives a considerably more perfect hardware platform for actualizing adaptable SoCs: the Xilinx presents the Zynq device which is the latest generation of Xilinx as a kind of all-programmable System-on-Chip (APSoC), that superbly catches its abilities. The internal architecture of the Zynq is presented in Figure 2.6 (Xilinx, 2018)**.**

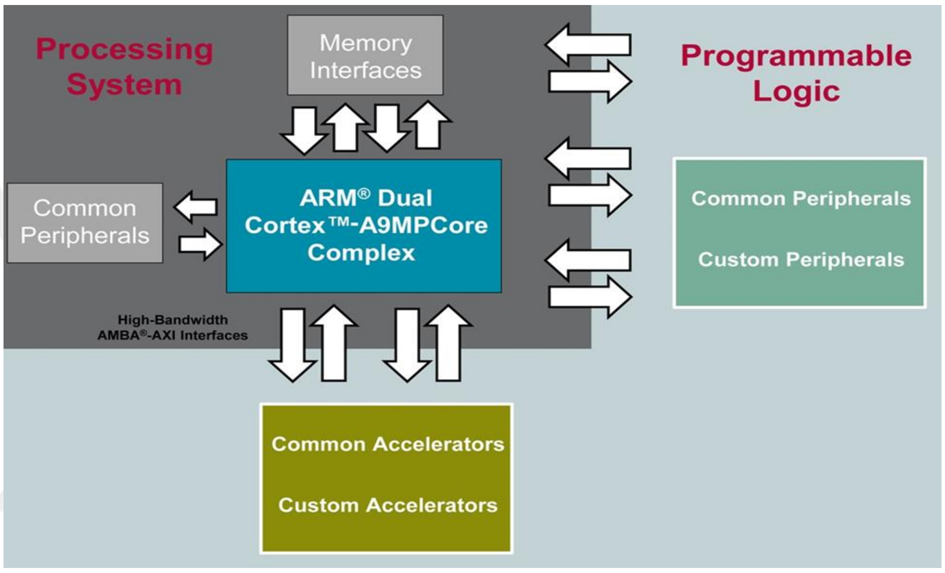


Figure 2.6. Internal Architecture of Zynq Device.

According to Figure 2.6, the internal architecture of the Zynq consists mainly of two parts: Processing System (PS) which comprises the dual-core ARM Cortex-A9 processors in addition to the multiple common peripherals and memory interfacing. and the second part is Programmable Logic (PL), that is equivalent to the traditional FPGA fabric (internal structure) and has a long range of general peripherals such as General-Purpose Input /Output (GPIO), different kind of memories, A/D converters, communications interfaces, and so on. The two parts of Zynq architecture can work together or independently.

2.6.2. Processing System (PS)

The Zynq processing system includes ARM processor with many associated hardware processing resources constructing the Application Processing Unit (APU), in addition to memory interfaces, clock generation, cache memory, group of peripheral interfaces, and high-performance interconnection (Xilinx, 2015). Figure 2.7 demonstrates the diagram of the Zynq PS (Xilinx, 2018).

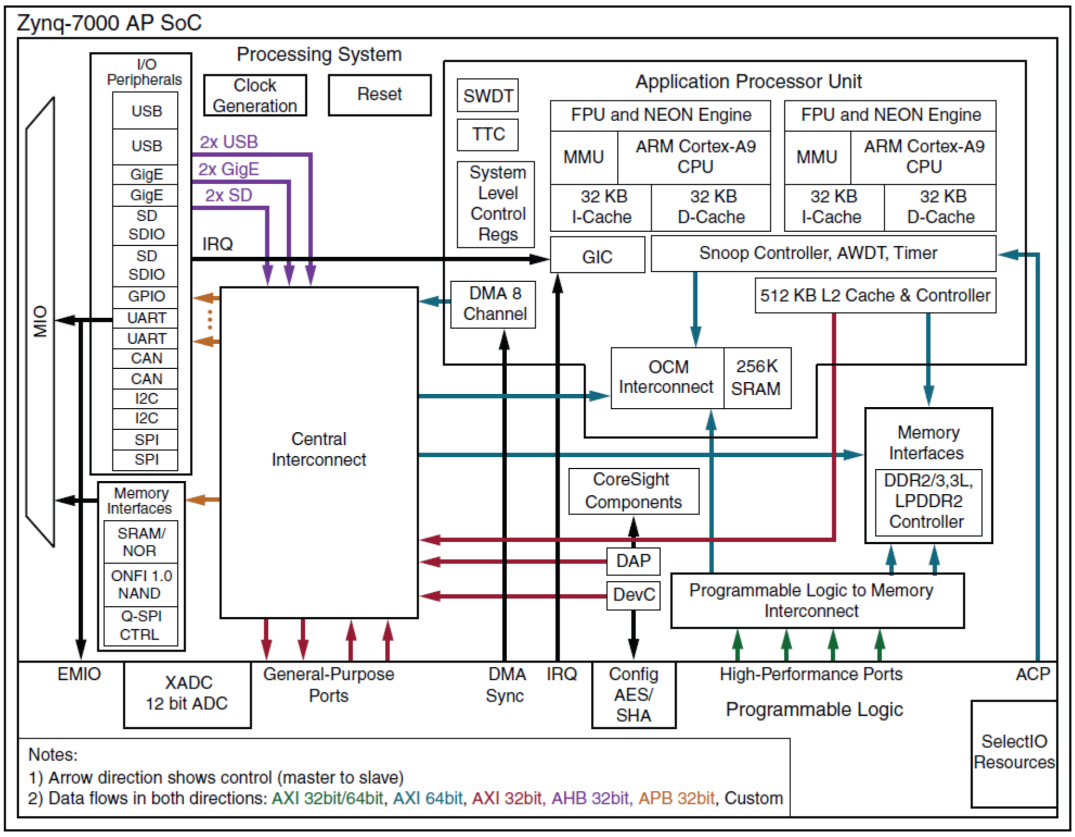


Figure 2.7. Diagram of the Zynq PS.

The frequency of the ARM processor is up to 1GHz, based on the series of the Zynq device. Each dual-core of ARM Cortex-A9 processor has two levels of cache memory: Level 1 and Level 2 caches. The size of Level 1 is 32KB which is used for data and instructions. This amount of cache is used to reduce the access time of the data and instruction that are frequently required and optimize the performance of the processor. While the size of Level 2 cache memory is 512KB which is shared between two core processors in addition to 256KB of on-chip cache within the APU.

On the other hand, the Software Development Kit (SDK) from Xilinx is used to configure the ARM processor which comprises all necessary tools and libraries to develop the processor’s functions.

Beside the ARM processor, the APU has NEON engine which utilize to provide Single Instruction Multiple Data (SIMD) mechanism to enhance the processing of multimedia (2D/3D graphics, video, audio, user interface, etc.) and Digital Signal

Processing (DSP) (Manual, 2015). Figure 2.8 explains the location of the NEON engine within the APU (Xilinx, 2018).

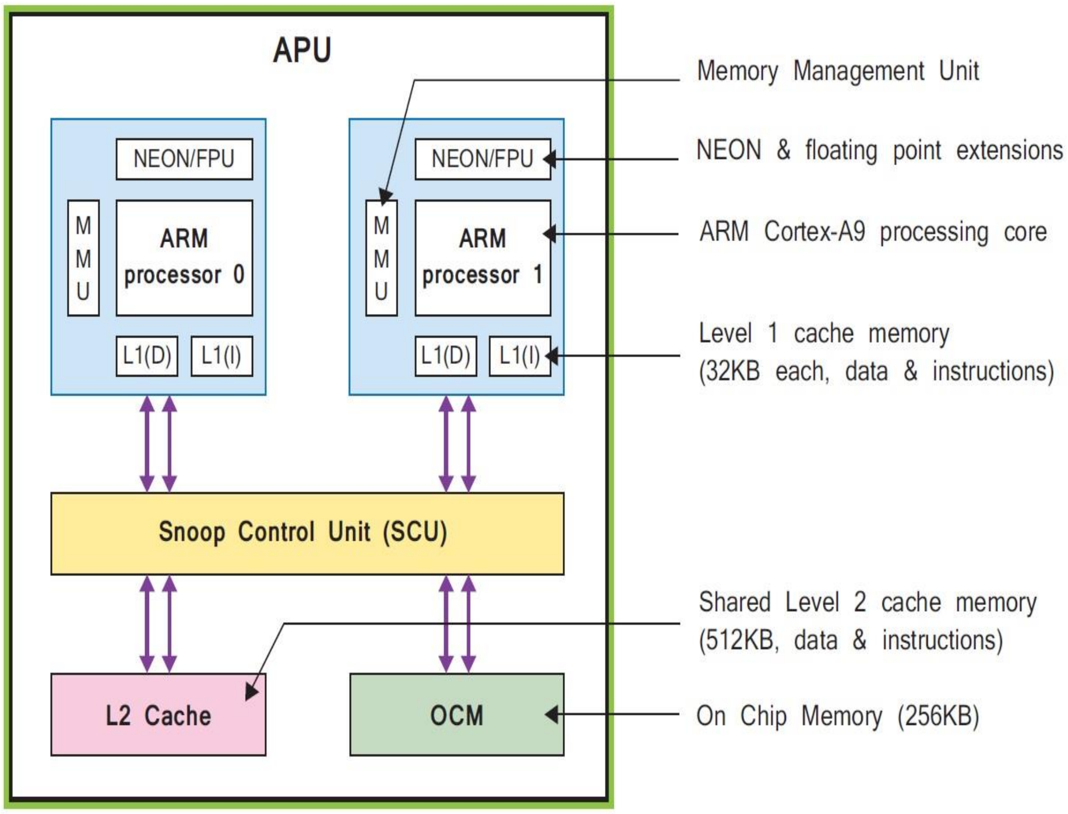

APU
NEON/FPU
MMU
ARM processor 0
L1(D)
L1(I)
NEON/FPU
MMU
ARM processor 1
L1(D)
L1(I)
Snoop Control Unit (SCU)
L2 Cache
OCM
Memory Management Unit
NEON & floating point extensions
ARM Cortex-A9 processing core
Level 1 cache memory (32KB each, data & instructions)
Shared Level 2 cache memory (512KB, data & instructions)
On Chip Memory (256KB)

Figure 2.8. Location of the NEON engine within the APU.

2.6.3. The External Interfaces of Processing System

There is a diversity of interfaces that are used by the Zynq PS to communicate the PL and the external devices as shown in Figure 2.7. According to the Figure 2.7, the Multiplexed Input / Output (MIO) are utilized to bind the PS part and external components. The MIO circuit is a flexible connectivity that consists of 54-pins which can be used to achieve different functions. The external connection also can be done by the extended MIO (EMIO) which is 192-signals (64 Inputs, 128 outputs) indirect connection from PS part to external environment through Zynq PL(Xilinx Inc. SDK, 2016). The MIO and EMIO demonstrated in Figure 2.9.

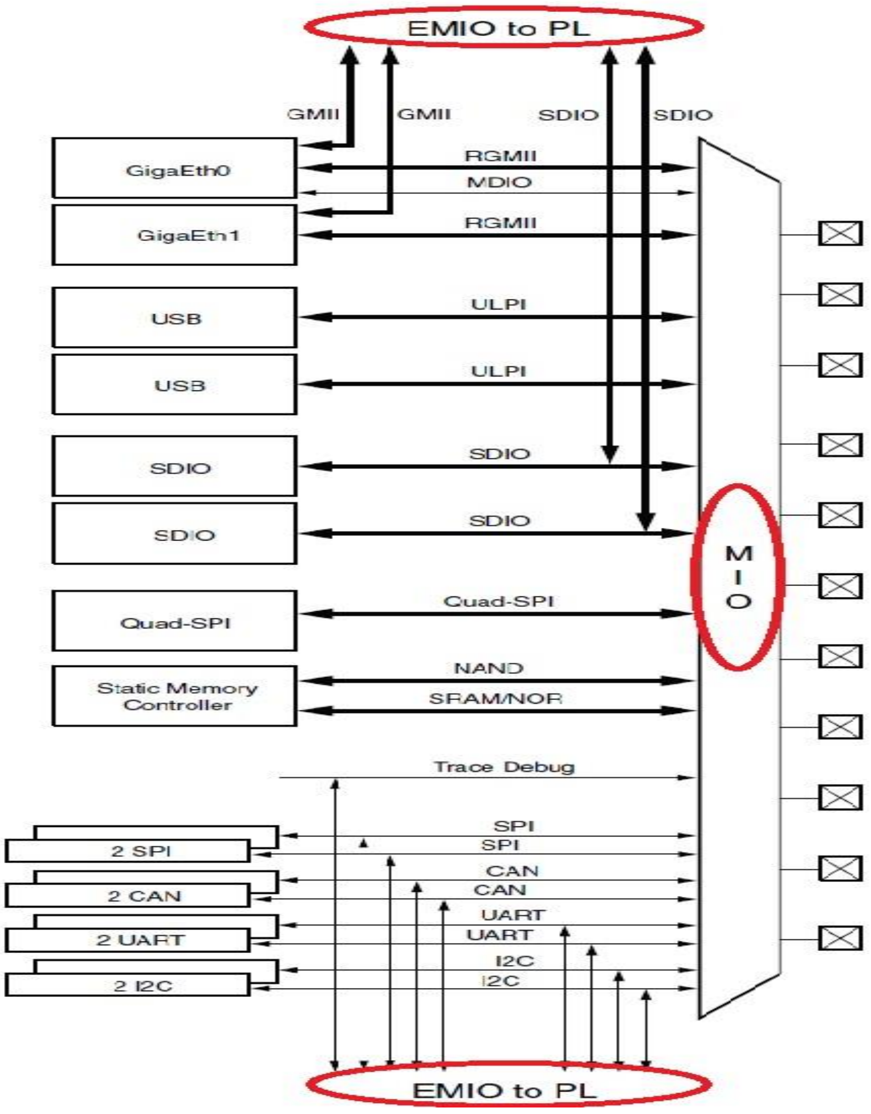
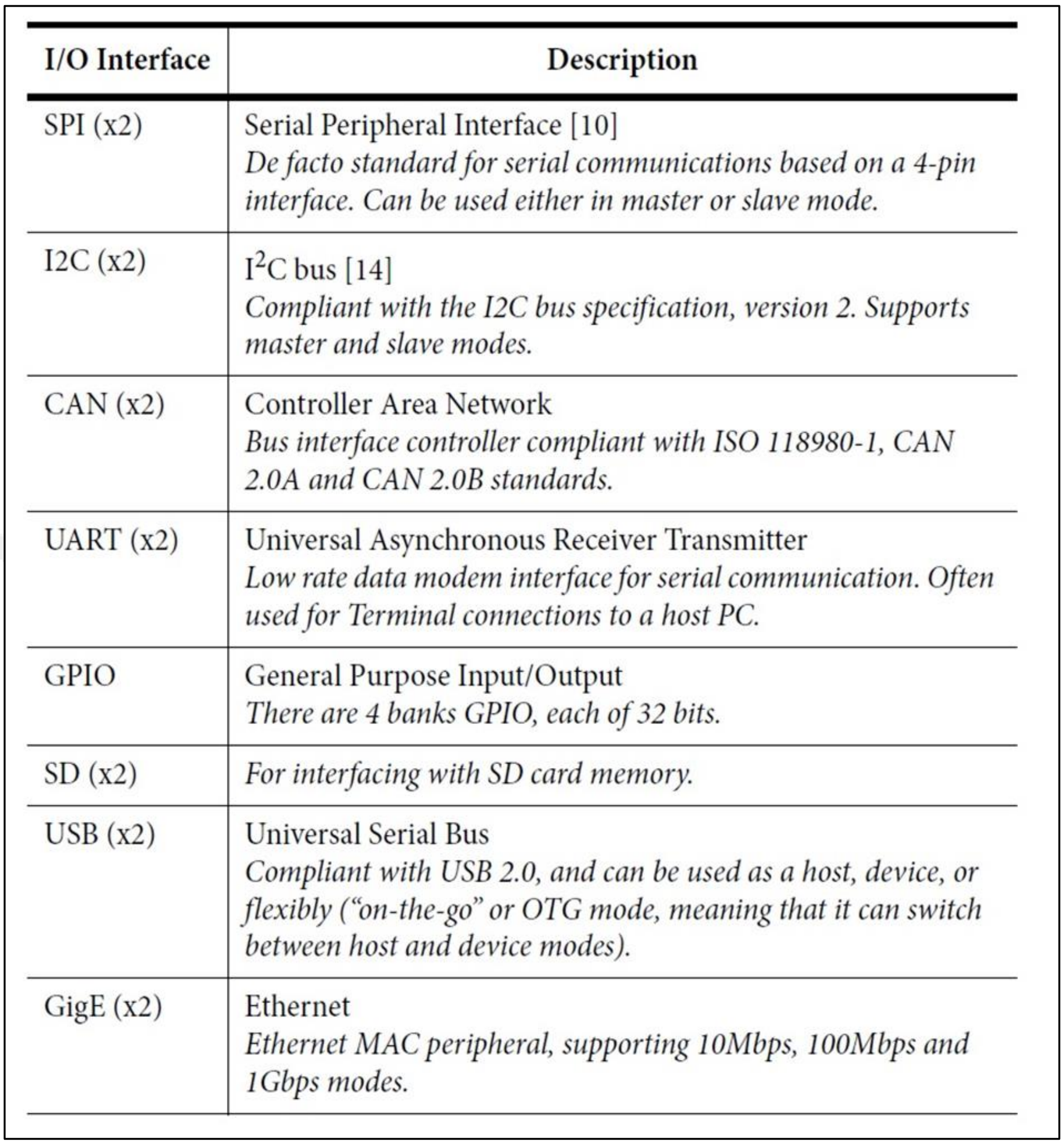


Figure 2.9. MIO and EMIO Interface.

The EMIO signals are normally utilized when any IP block core within the PL wants to access the PS part or more then 54-MIO is needed for any hardware design. The I/O ports within the MIO or EMIO consist of standard interface circuits, and (GPIO) that can be utilized for a long range of applications which comprise switches, and LEDs, etc. (Xilinx, 2018). The overall I/O peripheral is shown in Table 2.3 which include the standard name of each peripheral and its description.

Table 2.3. I/O Peripheral Description.



I/O Interface	Description
SPI (x2)	Serial Peripheral Interface [10] *De facto standard for serial communications based on a 4-pin interface. Can be used either in master or slave mode.*
I2C (x2)	I^2C bus [14] *Compliant with the I2C bus specification, version 2. Supports master and slave modes.*
CAN (x2)	Controller Area Network *Bus interface controller compliant with ISO 118980-1, CAN 2.0A and CAN 2.0B standards.*
UART (x2)	Universal Asynchronous Receiver Transmitter *Low rate data modem interface for serial communication. Often used for Terminal connections to a host PC.*
GPIO	General Purpose Input/Output *There are 4 banks GPIO, each of 32 bits.*
SD (x2)	*For interfacing with SD card memory.*
USB (x2)	Universal Serial Bus *Compliant with USB 2.0, and can be used as a host, device, or flexibly ("on-the-go" or OTG mode, meaning that it can switch between host and device modes).*
GigE (x2)	Ethernet *Ethernet MAC peripheral, supporting 10Mbps, 100Mbps and 1Gbps modes.*

2.6.4 Programmable Logic

The programmable logic (PL) is the second portion of the Zynq architecture. Figure 2.10 shows the main elements that can be included within this part. The PL part represents the general logic of the FPGA fabric. Generally, the Zynq PL is like any traditional FPGA device. It consists of general Input/Output Blocks (IOBs), Configurable Logic Blocks (CLBs), and programmable interconnection which is used to connect the internal parts of PL.

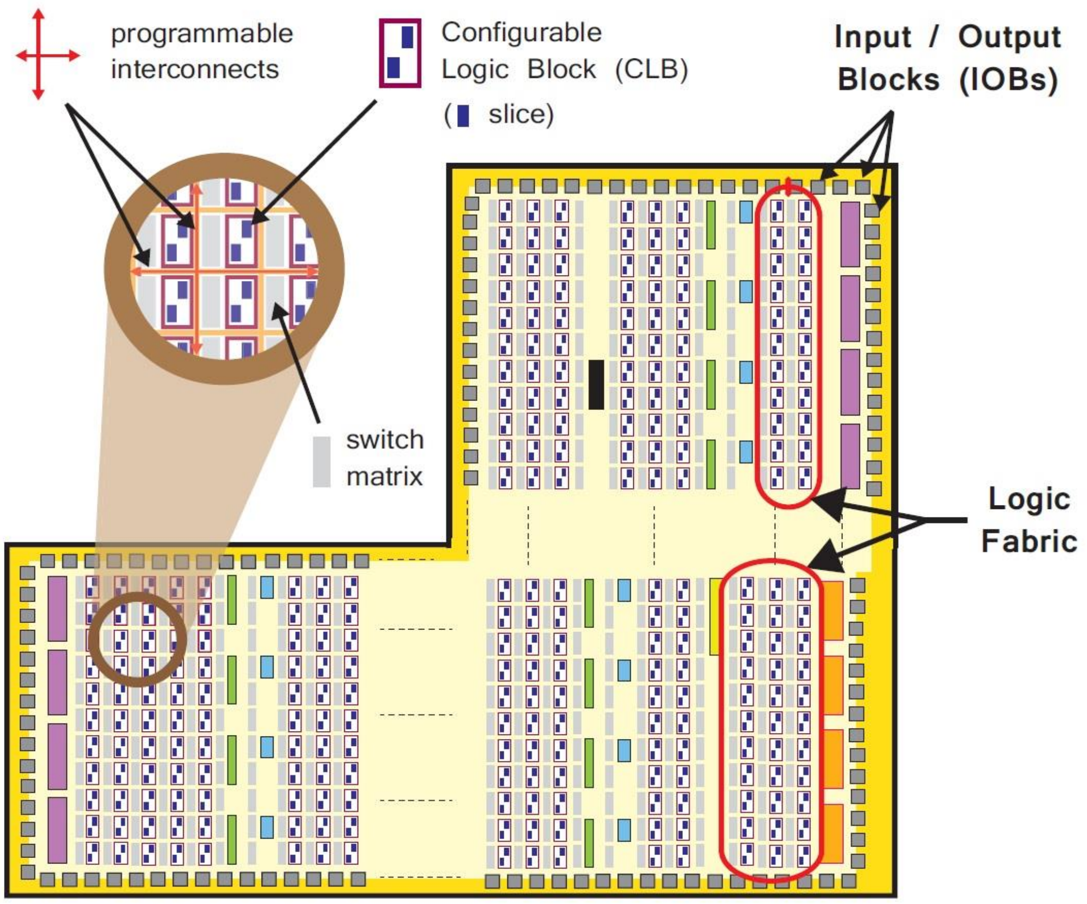

programmable interconnects
Configurable Logic Block (CLB) (▮ slice)
Input / Output Blocks (IOBs)
switch matrix
Logic Fabric

Figure 2.10. Zynq PL structure.

Based on Figure 2.10, all features of the PL can be abridged as shown below:

- Configurable Logic Block (CLB): The CLB is the basic logic element on the FPGA fabric. The general structure of the CLB consists of smaller logic element components, such as multiplexers, look-up tables (LUTs), and flip-flops. When connecting multiple CLB together based on switch matrix interconnection, the CLBs can be used to perform complex functions.

- Lookup Table (LUT): LUT is a group of logic circuits that are wired within the FPGA. It has the capability to implement many adaptable functions such as a small size of Random Access Memory (RAM) or Read Only Memory (ROM), shift registers, logic functions, etc.

- Flip-flop (FF): FF is a sequential logic circuit and normally works as a storage resource within the FPGA fabric. A one FF can be used as a binary register to keep the logic states during the FPGA clock’s cycles circuit.

- Slice: A sub-unit inside the CLB, which is used to implement the sequential and combinational logic circuits. The slice is constructed when a set of multiplexers, LUTs, and flip-flops are connected together.

- Switch Matrix: A switch matrix is a flexible programmable interconnection switch used to:
  1) connect the internal element of CLB,
  2) connect a group of CLB together,
  3) connect other resources of the PL together.

- Carry logic: Carry logic is a combination of multiplexer circuit and XOR gates are connected within the CLB. The main function of the carry logic is to spread the intermediate signals of the arithmetic circuits among adjacent slices.

- Input / Output Blocks (IOBs): IOBs are used to interfacing the elements of PL to the external physical peripherals. IOBs are placed around the circumference of the FPGA device.

  In general, there is no need for a deep knowledge of the internal architecture of the logic fabric by the designer, in most cases, all the design requirement from logic resources (LUTs, FFs, Multiplexers, gates, etc.) will be done automatically by the Xilinx tools accordingly.

2.6.5. Programmable Logic and Processing System Interfaces

As we mentioned, the Zynq architecture consists of PS and the PL parts. It is important to have the ability to use both of them in order to build complete and integrated systems. The key solution is the high flexibility configurable AXI interconnects which is a kind of bridge between the Zynq parts.in addition to other particular connections likes EMIO.

In this section, we will present the configurable interconnection between the Zynq PS and PL and show how they can be utilized. We start by explaining the AXI standard interconnection, which is the most used connection.

2.6.6. Advanced eXtensible Interface (AXI)

Advanced eXtensible Interface or shortly AXI4 is the fourth and current version of the ARM microcontrollers based on AMBA standard which was developed and released in 1996 to be used within ARM microcontrollers. Lots of devices and hardware cores are generated via third-party manufacturers are depended on this standard interface (Hinkelmann, Reinhardt, Varyani, and Glesner, 2008).

We are focusing on the Zynq device-base System-on-Chip which comprise the FPGA fabric. The AXI4 is a perfect reconfigurable interconnect technique that can be used within FPGA devices and the Xilinx is contributed strongly to develop it and used within its architectures(Reference et al., 2014). The Xilinx software tools starting with ISE® Design Suite 12.3 and up to the Vivado Design Suite are supported for using AXI interconnection.

Actually, there are three types of AXI4 buses, and each one can be implemented with a different bus protocol. The choice of the protocol-baed AXI bus depends on the connection's properties. Shortly, the types of AXI4 interfaces are (Hinkelmann et al., 2008):

- AXI4: Also known as memory-mapped protocol. This protocol provides a high performance and supports a data transfer up to 256 bytes per connection and usually use it for memory-mapped applications are requirements.

- AXI4-Lite: A very simple protocol and support single data transaction per connection. AXI4-Lite can be used for memory-mapped but the memory address must be provided at every starting connection by the master (read or write) signal.

- AXI4-Stream: high-performance, high-speed protocol and support a burst data transaction which its length is unknown. Typically, it is used for applications-based streaming of data. AXI4-Stream protocol works likes a unidirectional channel with a handshaking mechanism. The memory address is only determined at the first byte to be transferred.

2.7. WSN based FPGA background and literature review

Using the reconfigurable technique in the design of the architecture of sensor nodes (motes) can profit with regard to many aspects not only benefits from its high performance and flexibility but to but also prototype a new generation of processor architectures which can be used complex implementation. Recently, the Programmable Logic Devices (PLDs) such as FPGA devices are commonly utilized in the manufacturing of re-programmable sensor nodes. In this section, we will introduce the techniques which are used to design and implement the reconfigurable sensor node based on FPGA devices with some research works related with this approach. In general, there are three main techniques which are used to design and implement the reconfigurable sensor node based on FPGA devices (Wilder, Uzelac, Milenković, and Jovanov, 2008).

2.7.1. SoC-based FPGA prototype

Reconfigurability of the SoC-based FPGA prototype is utilized to design and implement various customized architectures of motes. New versions of FPGA devices have adequate resources to implement a full SoC architecture which include a microprocessor, different types of memories and lots of peripheral elements. Those SoC architectures can be done based on special tools from major FPGA vendors which back this feature like Vivado Design Suite from Xilinx or Embedded Design Suite from Altera.

Usually, these software have embedded tools which are utilized to debug, simulate and evaluate the hardware design. Furthermore, using these approaches provide perfect hints to optimize design tasks, performance, energy consumption. So, using FPGA prototype to design motes or any hardware system is the trustworthy method to guarantee that the system has correct functionally. Practically, the FPGA devices are chosen with large logic gates number to take advantage of its flexibility when evaluating the designing SoC architectures. Figure 2.11 shows this type of architecture.

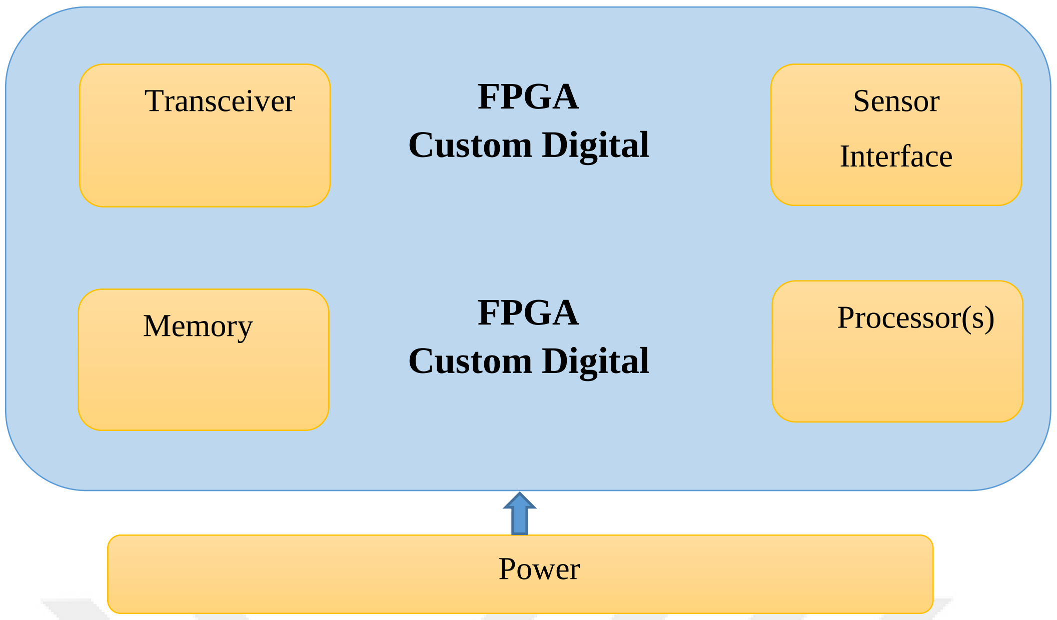


Figure 2.11. SoC-based FPGA prototype

Many works are done for WSN-based on this approach and implemented for different applications, for example, the authors in (Jiesheng Wei, Ling Wang, Feng Wu, Yibo Chen, & Long Ju, 2009) introduce an architecture based on SoC prototype and the system is targeted at Altera FPGADE2-70 kit.

This system has three main functional units which are: the interface circuit and the data acquisition unit, the Zigbee transceiver, and the data processing unit. All these units are handled by custom digital resources within FPGA fabric.

Another implementation for wireless sensor node can be found in(Liao, Singh, Khalid, & Tepe, 2013). The authors propose a parallel processing platform based on Xilinx Spartan3 FPGA. The proposed architecture can handle multi-events depend on its priority.

Machine monitoring and fault diagnosis platform-based on Xilinx Artix FPGA for WSN is suggested in (Bengherbia, Ould Zmirli, Toubal, and Guessoum, 2017). The node consists of a MicroBlaze soft processor as the main controller in addition to multiple hardware blocks such as AES block, nRF24L01 transceiver, and FFT block accelerometer.

2.7.2. FPGA as a standalone processing unit

FPGA as a standalone processing unit is an alternative design for sensor node based on re-programmable hardware (Figure 2.12). In this approach, MCU within the conventional sensor node is totally substituted by custom hardware-based FPGA and there is no need for any CPU within it. This hardware solution focuses on the implementation of the main functionalities of the core. So, the flexibility of the system and the energy efficiency will be affected negatively.

For this category, we can recognize between the designs using ASIC-based FPGAs prototyping and designs utilizing the custom hardware-based FPGA for sensor node in real-applications. In general, sensor nodes without microprocessor are suffering a generosity and focusing only on the target applications which make them convenient for lots of implementation.

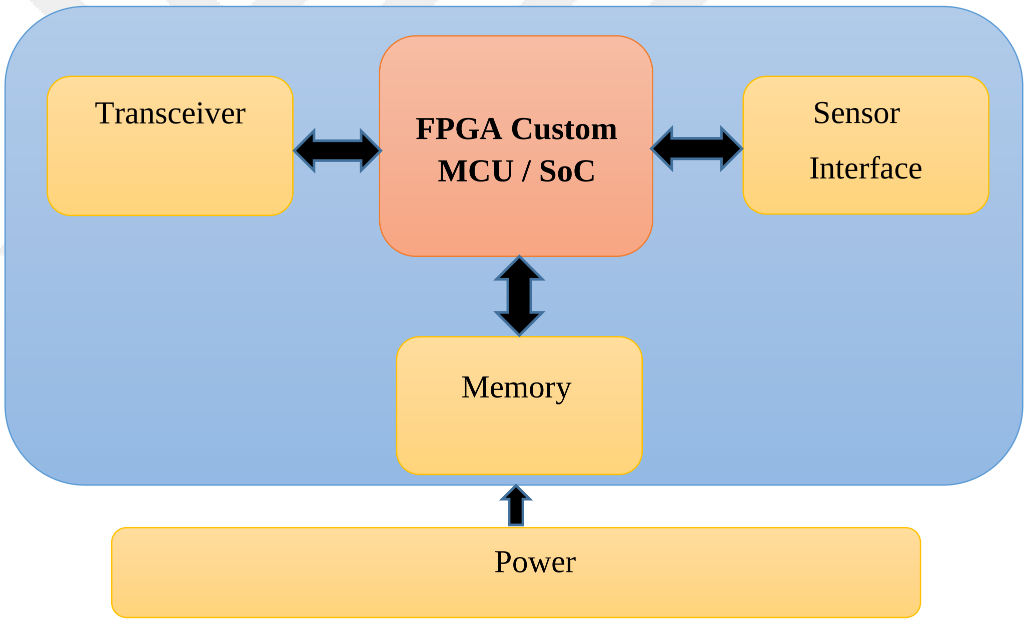


Figure 2.12. FPGA as standalone processing unit.

Many articles are proposed for WSN-based on this approach for instance the work in (Hinkelmann et al., 2008). The authors introduce a rapid prototype-based on reconfigurable FPGA for WSN. The system is designed and implemented based on Spartan3-2000 FPGA from Xilinx. The proposed platform consists of a radio transceiver, sensor module slots, a planar antenna, LEDs, and 4-rechargeable batteries. The processing part of the system and the interfacing to the external elements are handled by the FPGA.

The authors in (Muralidhar and Rao, 2008) introduce a wireless sensor node that can be used for fire detection and prediction. The Nios processor from Altera is used to design the sensor node with a re-programmable unit. The data acquisition process is performed by connecting the LM35DZ temperature to the FPGA hardware kit and the process of send/receive data is handled by PTR-4500 Bluetooth transceiver. Another similar work based on GSM communication can be found in(Suresh, 2018).

Dynamic re-configurable platform based on IGLOO FPGA is implemented in (Francois Philipp and Glesner, 2011) for WSN. In this work, Fast Fourier Transform (FFT) and Finite Impulse Response filter (FIR) are used to handle and process the sensing data by the system controller. According to the authors, the dynamic re-configurable approach can help to reduce the energy and execution time.

Partial reconfiguration is a technique used with the FPGA devices which divides the hardware system into dynamic and static parts. The dynamic part can be online re-configured while the remaining static part is still working without disturbing. So, the authors in (Meena and Krishna Prakash, 2014) proposed an architecture for a WSN based on partial reconfiguration Virtex-4 FPGA. The sensor node is comprised multiple different analog and digital sensors such as: temperature sensor, infrared sensor, smoke sensor, and photo detector in addition to another associated components like ADC converter, GPIO, and so on.

Another flexible partial re-configuration implementation for security and cryptography applications which can be employed for a wireless sensor node based on Xilinx Spartan 6 FPGA device is introduced in (Cardona, Lorente, and Ferrer, 2014). The authors propose a method that can be used to perform and synthesis the AES-256 and SHA-3 algorithms within the FPGA by are storing these algorithms in XC6SLX16-2CSG324 flash memory within Spartan 6. The Internal Configuration Access Port (ICAP) controller which is a kind of embedded access port is utilized to control the data flow to/from the flash memory.

The time delay between the controller and the flash memory and high-power consumption are the difficult challenges that appear when implementing this kind of security and cryptography algorithms for WSN. The FPGA prototype is not always convenient in WSN applications.

2.7.3. PLD as MCU co-processing unit:

This category is the last and most commonly used in the designing of the motes where the re-programmable devices are placed beside the MCUs in same mote architectures in order to accelerate the system's tasks as shown in Figure 2.13. In this case, the function of the re-programmable hardware is to implement all the tasks that cannot be done efficiently by the main CPU of the mote.

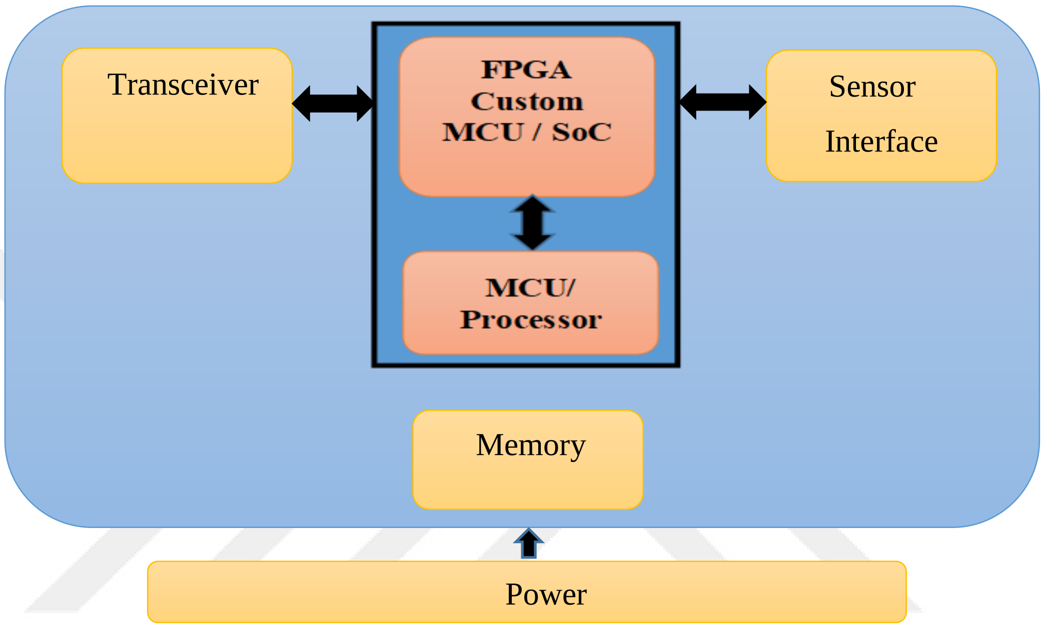


Figure 2.13. PLD as MCU co-processing

Most mote's architecture from this category are expected to be better compared with mote-based FPGA prototype and standalone processing-based FPGA unit in terms of performance, speed, and power consumption. Finally, the type of re-programmable devices will be used within the mote must be carefully chosen in terms of the number of logic resources in order to minimize the power consumption. So, works are done based on this methodology with and implemented for different applications. Table 2.4 summarizes some of these works based on Platform, MCU, Programmable devices, Sensors types, and Application.

Table 2.4. PLD as MCU co-processing.

Work-Year	platform	MCU	Programm-able devices	Sensors	Application
(Bellis et al., 2005) **2005**	NMRC sensor node	Spartan IIE	ATMega128L	-	General purpose
(Rahimi, Estrin, Baer, Uyeno, & Warrior, 2004) **2005**	Cyclops	ATmega128L	Xilinx XC2C256 CPLD	Image Sensor	Vision applications
(Kerhet, Magno, Leonardi, Boni, & Benini, 2007) **2007**	MicrelEye	AT40K MCU	FPSLIC	Camera OV7620	People detection
(Krasteva, Portilla, Carnicer, De La Torre, & Riesgo, 2008) **2008**	Cookie	ADuC841	Xilinx Spartan3 XC3S200	Humidity, Light, Infrared	Environment
(Imran, Khursheed, O'Nils, & Lawal, 2010) **2010**	μc FPGA	SENTIO32	Xilinx Spartan6 IGLOO	Image sensor	magnetic particles detection
("PowWow: Power Optimized Hardware and Software FrameWork for Wireless Motes," 2018) **2010**	PowWow	MSP430	FPGA IGLOO 125	-	General purpose
(W. Liu, Bienstman, Jooris, Yaron, & Moerman, 2012) **2011**	Liu et al.	TelosB	Xilinx Spartan-3A and 3E	-	Experimental platform
(Yuan, Qiu, Gao, Tong, & Yang, 2012) **2012**	Yuan et al.	MSP430	AN231E04 FPAA	Sensor board	Self-healing paradigms

(Kelly, Rumberg, Graham, & Kulathumani, 2013) **2013**	Kelly et al.	TelosB	CPLD 2 FPAA	Gyroscope, Microphones	General purpose
(Szilvási, Babják, Völgyesi, & Lédeczi, 2013) **2013**	Marmote SDR SmartFusion	ARM	IGLOO FPGA	-	General purpose
(Lopez, Valverde, De La Torre, & Riesgo, 2014) **2014**	HiReCookie	Spartan-6	MicroBlaze	Image sensor	Multimedia applications
(Fularz, Kraft, Schmidt, & Kasi, 2015) **2015**	Zynq	ARM cortex A9	Artix7 xc7z020-clg484	Image sensor	Video surveillance & inspection
(Zhai, Ali, Amira, & Bensaali, 2016) **2016**	Zynq	ARM cortex A9	Artix7 XC7Z010T 1CLG400	Gas sensor	Gas monitoring
(Gomes, Pinto, Salgado, Tavares, & Cabral, 2017) **2017**	Smart RF06EB	ARM Cortex-M3	IGLOO FPGA	-	IoT application
(Prakash & Surjith, 2017) **2017**	Krishna et al.	MSP430 F5438	Spartan 3AN FPGA	Sensor board	Smart Grids

CHAPTER 3
ARTIFICIAL NEURAL NETWORKS AND FUZZY SYSTEM

3.1. Introduction

Artificial neural network (ANN) is an information technology (IT) and processing system of software and /or hardware based on the architecture and the functionality of the biological human neural networks (Fausett and Fausett, 1994). ANNs are non-linear modeling tools that can be utilized when the inputs and outputs patterns have complex relationships. It can be described by:

The Architecture which is the type of connection between its neurons (nodes), the training Algorithm which is the methodology of calculating the associated weight with each node, and finally, the types of its activation functions (Fausett & Fausett, 1994).

There are many differences between biological and artificial networks. In general, "artificial neural networks" refer to the software-based algorithm and hard-wired structure rather than physical biological system (Konar, 1999; Mehrotra, Mohan, and Ranka, 1997; Dreyfus, 2005).

3.2. How Artificial Neural Networks Work

Any artificial neural network comprises an enormous number of processors nodes working in parallel and distributed in layers. The first layer accepts the input data like an optic nerve in human. Each consecutive layer receives its input from the output of the layer preceding it. The last layer of the artificial neural produces the system output.

The knowledge is distributed on all processing nodes through the layers of the network including any origin pre-programmed or modified rules. The layers are strongly interconnected, that means each node in each layer is connected to all nodes in consecutive layer. The output layer could have more than one output node depending on the application for which the artificial neural network is used.

3.3. Elements of Neural Networks

The basic structure of an artificial neuron is shown in Figure 3.1 which demonstrates the basic components of an ANN. It comprises three basic elements which include weights, summation function, and a single activation function(Fausett and Fausett, 1994 ; Mehrotra et al., 1997).

3.3.1. Weighting Factors:

The values of W_1, W_2, W_3 ,…..,W_n in Figure 3.1 are weight factors which are linked with each node to calculate the strength of input $X = [x_1\ x_2\ x_3 \ldots, x_n]^T$. The associated weight with each node is multiplied by corresponding input data in the form ($X^T * W$). The weight factors are adaptive coefficients inside the artificial network which calculate the strength of the input data. In other word, the intensity of any input connection is measured by those adaptive coefficients. They can be adjusted in order to respond to different training sets and based on its architecture.

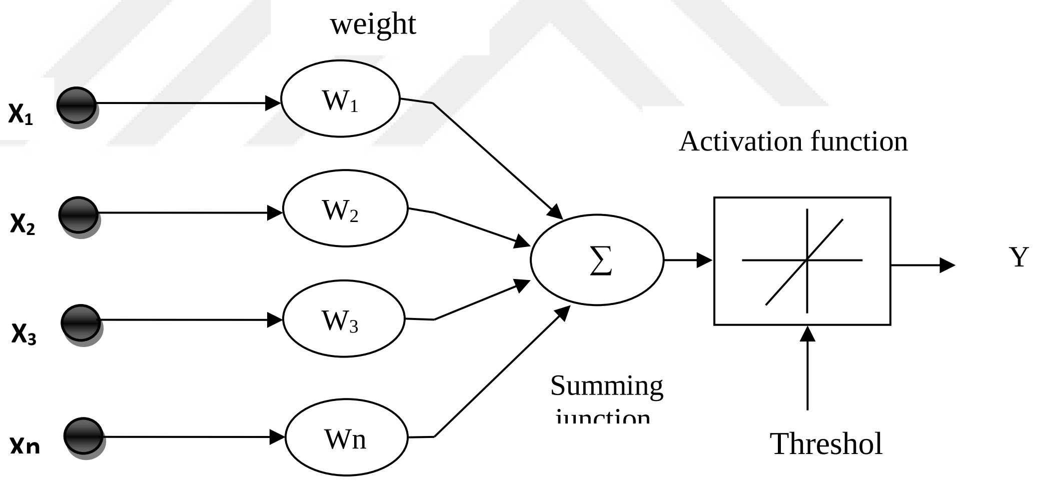


Figure 3.1. Basic Elements of an Artificial Neuron.

3.3.2. Summation Function

This function is the first level in a data processing. It is also known as a combination function. The summation function is using to compute the summation of the inner product of all inputs data (x_1, x_2, x_3… x_n) and associated weights vectors (w_1, w_2, w_3 . . . w_n). Mathematically, we can use a simple summation function to represent the multiplication between each input data vector and the corresponding weight vector. Eq. 3.1 shows this relationship.

$$y = f(x, w, b) = b + \sum x_i * w_i, \quad i = 1,2,3, \qquad (3.1)$$

The Eq. 3.1 also includes a net bias (b), that decreases or increases the net input value to the activation function.

3.3.3. Common Activation Function

The Activation Function is utilized to define the output type of the neural network. There are many types of activation function which can be used in any neural network based on the type of application which is realized by the network. In this subsection, we will present four of the common activation functions which are used in most linear or non-linear algorithms (Sivanandam, Sumathi, and Deepa, 2007;Tettamanzi and Tomassini, 2013; Sumathi and Paneerselvam, 2010).

3.3.3.1. Linear Function

The output of the neuron model based on this type of activation function is the same to the network input data**.** The linear activation function is demonstrated in Figure 3.2.

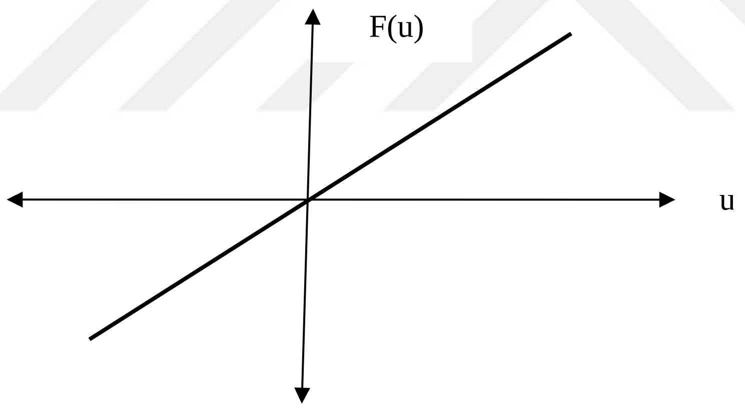


Figure 3.2. Linear Activation Function.

The formula in Eq. 3.2 describes the linear activation function:

$$y = F(x) = \alpha . x \qquad (3.2)$$

where α is the slope of the linear function. If the value of α equals 1, then the activation function is known as an identity function. As we mentioned above, the output (y) of the identity activation function is equal to the net input signal (x).

3.3.3.2. Threshold Function

A threshold function or hard-limiter activation function. It can be used in two binary or bipolar forms as demonstrated in Figures (3.3) and (3.4), respectively. Eq. 3.3 describes the output of a binary activation function:

$$y = f(x)\begin{cases} 0 & \text{if} \quad x < \theta \\ 1 & \text{if} \quad x \geq \theta \end{cases} \tag{3.3}$$

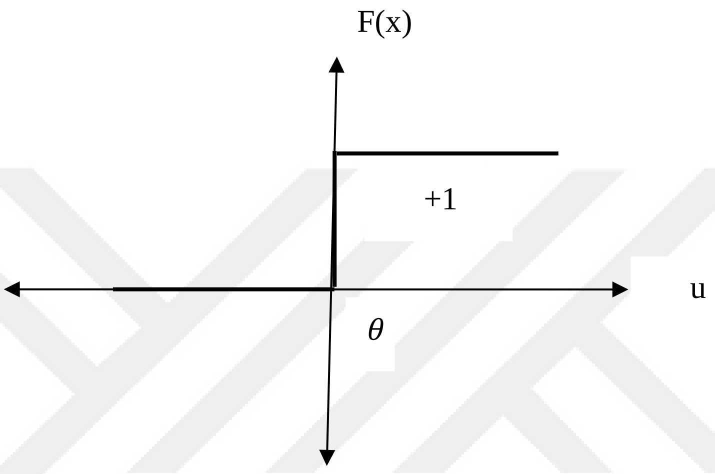


Figure 3.3. Binary Threshold Activation

while the bipolar function can be written according to the Eq. 3.4:

$$y = f(x)\begin{cases} 1 & \text{if} \quad x \geq \theta \\ -1 & \text{if} \quad x < \theta \end{cases} \tag{3.4}$$

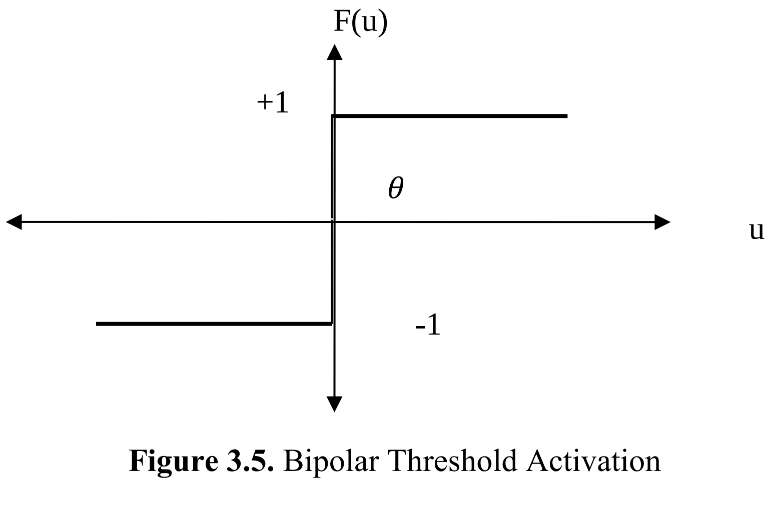


Figure 3.5. Bipolar Threshold Activation

3.3.3.3. Piecewise Linear Function

A piecewise linear activation function is a function comprises multiple linear segments which are defined according to specific periods**.** The piecewise linear function is known as saturating linear function which can be used in binary or bipolar form**.** Figure 3.5 shows the mathematical form for a symmetric piecewise linear activation function and described in Eq. 3.4:

$$y = f(u) = \begin{cases} -1 & \text{if} \quad u < -\theta \\ u & \text{if} \quad -\theta \leq u \leq \theta \\ 1 & \text{if} \quad u > \theta \end{cases} \qquad (3.4)$$

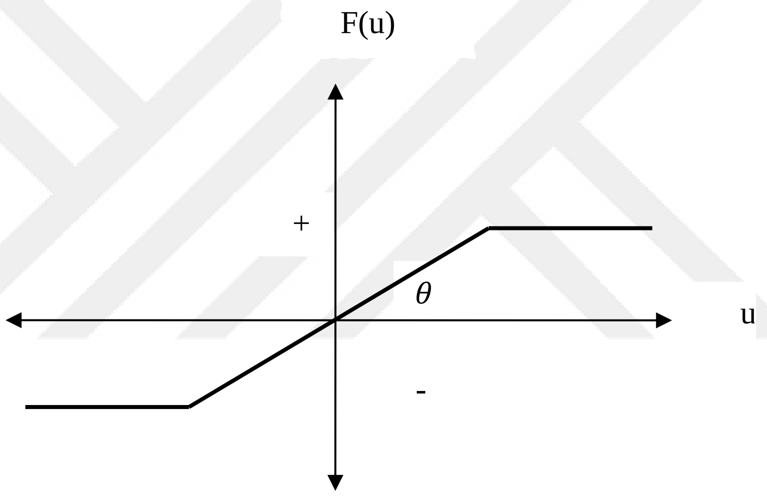


Figure 3.7. Piecewise Linear Activation Function.

3.3.3.4. Sigmoid Function

This type of activation function is the most common utilized function within neural networks. The sigmoid function provides a perfect equilibrium among linear and non-linear behaviors. Hyperbolic tangent activation function is an example of a sigmoid function which can be defined by Eq. 3.5 and demonstrated in Figure 3.6.

$$y = \tanh(u) \qquad (3\text{-}5)$$

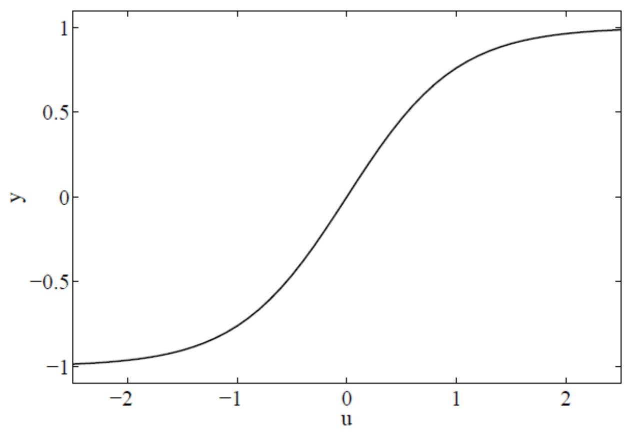


Figure 3.8. Tangent Activation Function.

3.4. Training of Neural Networks

To start training, first, the structure of the neural network must be specified to be suitable for a specific application, second, all internal initial weights of the network are randomly picked. Then the training process will begin. Typically, two approaches of training any neural network are most commonly used: supervised and unsupervised methods.

3.4.1. Supervised Approach

Supervised training approach includes a mechanism to provide the network with a set of inputs and desired outputs. For instance, to construct a neural network to recognize the face of clowns, a large number of pictures of clowns, non-clowns, animal faces, human faces with masks and so on must be given in initial training phase. Each input data is associated with the specific identification, such as clowns' names, not clowns, animals, and so on. The internal initial weightings start to adjust themselves to reach a better performance. Most of the artificial networks utilize supervised training approach. In other word, the training set in this approach which includes the inputs and the desired target are provided by the user. The neural network then starts training the input data and compares its outputs result versus the desired targets. The amount of error is calculated in order to modify the overall weights that control the neural network. This process is repeated continuously over the time until the weights reach to the situation level (Fausett and Fausett, 1994; Galushkin, 2007).

In some cases, the training process doesn't complete in some artificial network algorithms. This case could appear when the provided input data doesn't include enough information that is needed by the desired target. As a result, the artificial networks don’t have the ability to converge. So, the data should be divided into multiple folds and specify some of these folds in order to test the performance of the network.

The designer should review the overall network when the problem can't besolved based on the current configuration. The review process includes the type of the network architecture, the number of layers and nodes within each layer, the type of activation function, input and output data, the initial weights, and training method. All these review elements are required to build a successful artificial network.

The type of training algorithm is an important aspect the designers should consider. There are many training algorithms utilized to adapt the network weights during the training stage. The back-propagation algorithm is an example of one of the most common techniques which is used with the supervised approach.

3.4.2. Unsupervised Approach

The unsupervised training is the other type of training technique. In this approach, the artificial network is only provided with input data without desired targets. The type of learning algorithm which is used in the artificial network will construct the features from input data and then divide them into different groups. This is usually known as self-organization technique or self-adaption method.

Tuevo Kohonen from Helsinki University of Technology is one of the famous leading researchers who worked to develop a self-adaption artificial network which learns without the advantage of knowing the correct answer. In general, these kinds of networks comprise only one layer with multiple connection nodes. The weights associated with each connection must be initialized and the provided input data should be normalized in order to determine the winner-takes-all based on self-adaption technique(Fausett and Fausett, 1994; Anderson and McNeill, 1992).

The self-adaption to the real environment is the promising technique that activates the robot’s science fiction to self-learn as they face new situations and environments where same group of training sets could not find. An example of this unexpected situation includes military actions, especially where a new combat technology or new weapons are encountered.

3.5. Types of Neural Networks

There are several kinds of Artificial Neural Networks (ANN)–based on its architecture and function(Jain and Martin, 1998;Graupe, 2013). Only most common and frequently used will be considered.

3.5.1. Feed-Forward ANN

This type of artificial neural network is a first and basic type of neural network. The input data moves in one direction from the input layer to the output layer through hidden layers (if present). There are no loops and feed-back are in this kind of ANN as shown in Figure 3.7 which consists of multi-layer feed-forward ANN. The feed-forward neural networks are originally utilized for supervised learning technique. In general, this type is used for simple applications such as recognizing input patterns and simple data classification.

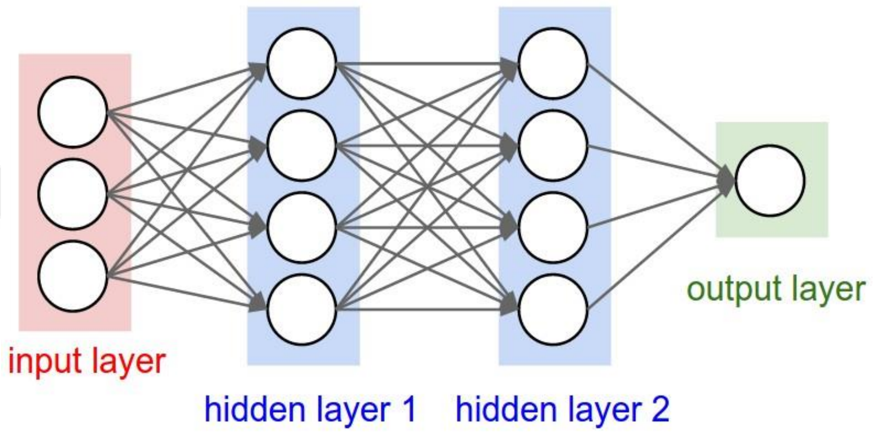


Figure 3.9. Multi-layer Feed-Forward ANN.

3.5.2. Feed-back ANN

The structure of feed-back ANN (or recurrent network) is like feed-forward ANN but information about the output sends back into the previous hidden layers to obtain best performance results as demonstrated in Figure 3.8. Generally, feed-back network is adapted network so, their 'state' is modifying constantly until it arrives an equilibrium point. The feed-back ANN is suited to realize optimization problems such as nonlinear problems and system error corrections.

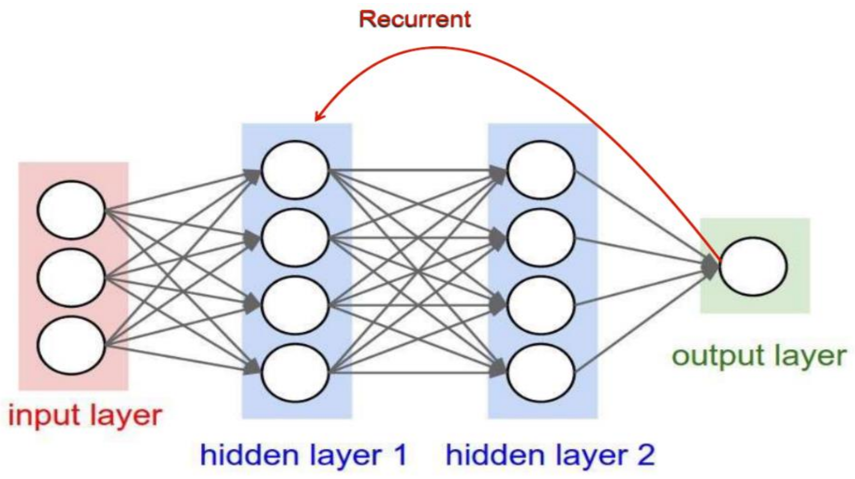


Figure 3.10. Multi-layer Feed-back ANN.

3.5.3. Classification-Prediction ANN

It is a type of feed-forward ANN. Usually, the classification-prediction artificial neural network is used for data-mining applications. This kind is trained to distinguish particular data patterns and classify them into qualitative groups. Figure 3.9 shows that an example for multi-layer classification network.

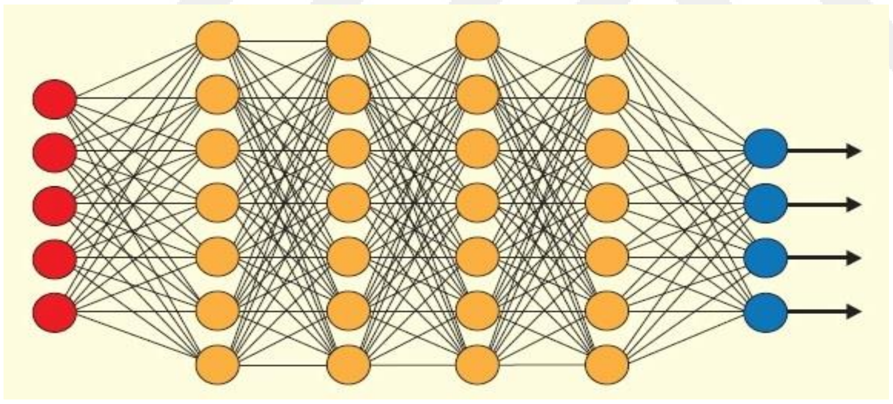


Figure 3.11. Multi-output Feed- Forward ANN.

3.6. Feedforward Back-Propagation

The back-propagation (BP) algorithm was developed in the early 60s by multiple researchers (David E. Rumelhart, James McClelland, Geoffrey E. Hinton, and Ronald J. Williams). The (BP) algorithm is a multi-layer feedforward neural network and is one of the most commonly used in the application of neural networks. The BP algorithm can be utilized to train and keep a great deal of non-linear and complex relations between input and output data patterns. The steepest descent technique is used as a learning rule in back to adjust the values of the network weight and threshold to get the minimum square of the error between the target and the network output.

The typical (BP) algorithm has an input layer, one or more hidden layers, and an output layer. The number of hidden layers depends on the complexity of the application and there is no theoretical constraint about the number of hidden layers which are utilized within its architecture but in general, one or two hidden layers are widely used(Amari, 2003). From the input to the output layer, the information will flow during the Recall process. The Recall is the operation of transferring the input data through a trained network and obtaining the net output (feed-forward phase). While Back-propagation of error is only used when the algorithm is learning a training data set. Figure 3.10 shows an example of a typical (BP) architecture with two hidden layers.

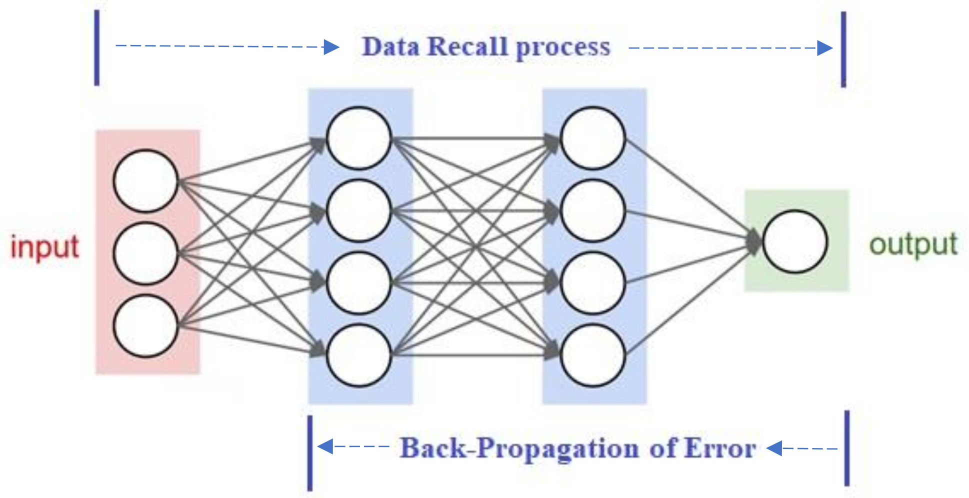


Figure 3.12. Example of Typical (BP) Architecture.

3.6.1. Learning in Back-Propagation Algorithm

The learning in back-propagation algorithm can be divided into two phases as describe below (Williams and Zipser, 1995; Dalgleish et al., 2007):

- **Forward phase (Recall process):** The input data is propagated over the network layers (input layer, hidden layer, and output layer). All weights and thresholds which are applied to the network are maintained constant during the forward of operating phase. In case, that the foreseeable output of the neural network cannot be achieved correctly, so the back- propagation of error can be started at this point. The forward propagation phase can be expressed by the following equations:

The output of the of hidden layer:

$$y_j = f\left(\sum_i w_{ij} - \theta_j\right) = f\left(net_j\right) \qquad (3.5)$$

$$net_j = \sum_i w_{ij} x_i - \theta_j \qquad (3.6)$$

And the output at the output layer:

$$z_l = f\left(\sum_j v_{lj} y_j - \theta_l\right) = f(net_l) \qquad (3.7)$$

$$net_l = \sum_j v_{lj} y - \theta_l \qquad (3.8)$$

While the error at the output layer can be achieved by the following equation:

$$E = \frac{1}{2}\sum_l (t_l - z_l)^2 = \frac{1}{2}\sum_l (t_l - f(\sum v_{lj} y_j - \theta_l))^2$$

$$= \frac{1}{2}\sum_l (t_l - f(\sum_j (v_{lj}\, f(w_{ji} x_i - \theta_j) - \theta_l))^2 \qquad (3.9)$$

Where:

i,j: 1,2,3,

x_i: The input data,

y_j: The output of hidden layer,

z_l**:** The output of the output layer,

w_{ji}: The weight between the input and hidden layer,

v_{lj}**:** The weight between the hidden and output layer,

t_l**:** The expected value of the output node is,

*f(net)***:** is the active function.

- **Back-propagation of error:** The error at the output layer can be calculated by the diversity between the real output (or target) and the calculated output of the algorithm; the calculated error in back-propagation phase is propagated from the output layer to all hidden layer. During this phase, all weights and thresholds of network are modified based on the value of the error. This modification of weights and thresholds is repeated continuously until target be closer to the output of the algorithm. The gradient descent technique is applied to modify the value of the weights at all layers as demonstrated in the following equations:

1. **Modification of Weight Value**

The first step is to find derivation of output based on the error function

$$\frac{\partial E}{\partial v_{lj}} = \sum_{k=l}^{n} \frac{\partial E}{\partial z_k} * \frac{\partial z_k}{\partial v_{lj}} = \frac{\partial E}{\partial z_l} * \frac{\partial z_l}{\partial v_{lj}}$$

Where *E* **(**error**)** is a function including multiple ***z$_k$***, but just one of ***z$_l$*** is linked with ***v$_{lj}$*** and all other ***z$_k$*** are independent, as shown below.

$$\frac{\partial E}{\partial z_l} = \frac{1}{2}\sum_{k}[-2(t_k - z_k) * \frac{\partial z_k}{\partial z_l}] = -(t_l - z_l)$$

$$\frac{\partial z_l}{\partial v_{lj}} = \sum_{k=l}^{n} \frac{\partial z_k}{\partial net_l} * \frac{\partial net_l}{\partial v_{ij}} = f'(net_l) * y_j$$

In this way,

$$\frac{\partial E}{\partial v_{lj}} = -(t_l - z_l) * f'(net_l) * y_j$$

Then, we will assume that the error of output node is:

$$\delta_l = (t_l - z_l) * f'(net_l) \qquad (3.10)$$

So,

$$\frac{\partial E}{\partial v_{lj}} = -\delta_l * y_j \qquad (3.11)$$

Based on the error function, the deviation of hidden layer can be done using the following formula:

$$\frac{\partial E}{\partial v_{ji}} = \sum_l \sum_j \frac{\partial E}{\partial z_l} * \frac{\partial z_l}{\partial y_j} * \frac{\partial y_j}{\partial w_{ji}}$$

Also, the E function has multiple z_l; but just one is targeted at specific w_{ji}, and it is corresponding to single y_j, and regarding to all z_l as explained in below:

$$\frac{\partial E}{\partial z_l} = \frac{1}{2}\sum_k [-2(t_k - z_k) * \frac{\partial z_k}{\partial z_l}] = -(t_l - z_l)$$

In this way:

$$\frac{\partial E}{\partial w_{ji}} = -\sum_l (t_l - z_l) * f'(net_l) * v_{lj} * f'(net_j) * x_i$$

$$= -\sum_l (\delta_l v_{lj}) * f'(net_j) * x_i$$

Assume that the error of hidden layer is:

$$\delta'_j = f'(net_j) * \sum_l (\delta_l v_{lj})$$

In this way:

$$\frac{\partial E}{\partial w_{ji}} = -\delta'_j \, x_i$$

The modification of weight $\boldsymbol{\Delta v_{lj}}$ and $\boldsymbol{\Delta w_{ji}}$ is in proportion to the error ratio and decline over the gradient, the relationship explaining the modification of all weights in both hidden layers and output layer is expressed below:

$$\Delta v_{lj} = -\alpha \frac{\partial E}{\partial v_{lj}} = \alpha \delta_l y_j \tag{3.12}$$

while the equation below shows that the modification of the weight among the input layer and hidden layers is expressed as shown below:

$$\Delta w_{ji} = -\alpha' \frac{\partial E}{\partial w_{ji}} = \alpha' \delta'_j x_i \tag{3.13}$$

$$\delta'_j = f'(net_j) * \sum_l (\delta_l v_{lj}) \tag{3.14}$$

In above formula, the term (∑ **δ v**) in the hidden layer shows that the error **δ_l** of output node **z_l** is back-propagated over the weight value ***v_{lj}*** to the all ***y_j*** nodes to become the error of hidden layer.

2. **Modification of Threshold Value**

The threshold value θ is also adjusted during the training phase and it is important to be modified when the weight value is modified; the method which utilize is the same as that applied in the weights adjustment. As we did in the previous section, we will start to derivation the threshold value at output layer by error function as shown below:

$$\frac{\partial E}{\partial \theta_l} = \frac{\partial E}{\partial z_l} * \frac{\partial z_l}{\partial \theta_l}$$

In this formula:

$$\frac{\partial z_l}{\partial \theta_l} = \frac{\partial z_l}{\partial net_l} * \frac{\partial net_l}{\partial \theta_l} = f'(net_l) * (-1)$$

In this way, the equation expressing the adjustment of threshold is

$$\Delta\theta_l = \alpha \frac{\partial E}{\partial \theta_l} = \alpha\delta_l$$

Namely:

$$\theta_l(k+1) = \theta_l(k) + \Delta\theta_l = \theta_l(k) + \alpha\delta_l$$

Now, the threshold derivation of the hidden layer based on the error function can be expressed as shown below:

$$\frac{\partial E}{\partial \theta_j} = \sum_l \frac{\partial E}{\partial z_l} * \frac{\partial z_l}{\partial y_j} * \frac{\partial y_j}{\partial \theta_j}$$

So, in this formula:

$$\frac{\partial y_j}{\partial \theta_j} = \frac{\partial y_j}{\partial net_j} * \frac{\partial net_j}{\partial \theta_j} = f'\left(net_j\right) * (-1) = -f'\left(net_j\right)$$

In this way:

$$\frac{\partial E}{\partial \theta_j} = \sum_l (t_l - z_l) * f'(net_l) * v_{lj} * f'\left(net_j\right) = \sum_l \left(\delta_l v_{lj}\right) * f'\left(net_j\right) = \delta'_j$$

Now, the equation that expressing the threshold modification is:

$$\Delta\theta_j = \alpha' \frac{\partial E}{\partial \theta_j} = \alpha' \delta'_j$$

Namely:

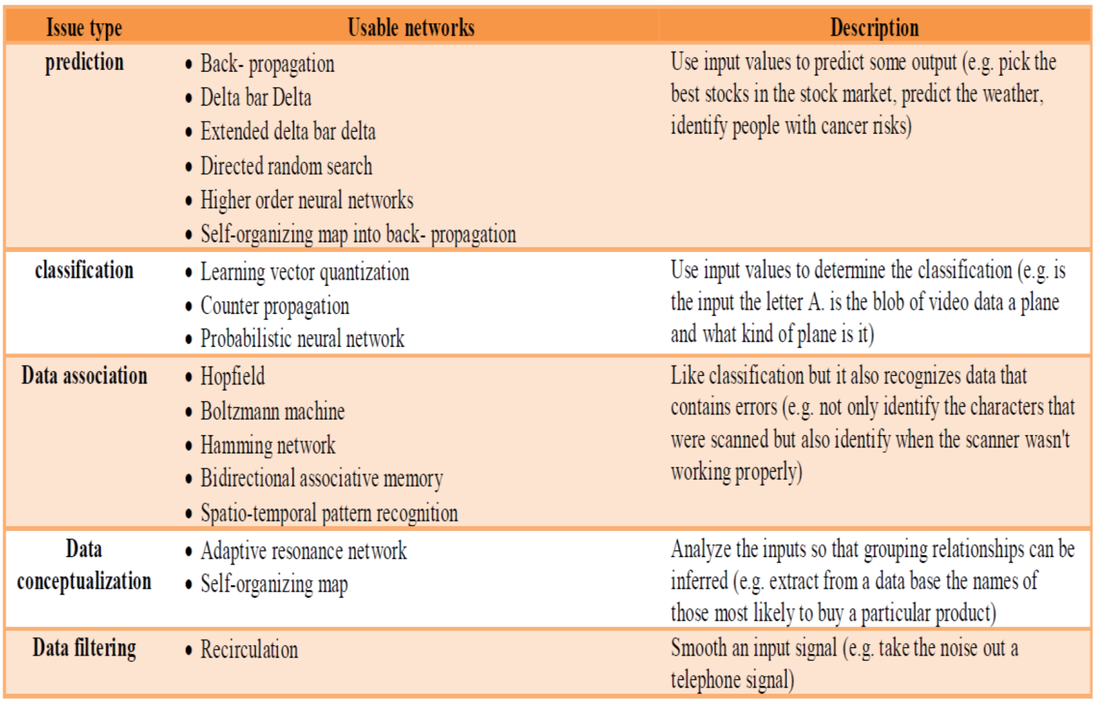
$$(k + 1) = \theta_j(k) + \Delta\theta_j = \theta_j(k) + \alpha' \delta'_j \qquad (3.15)$$

3.7. Network Selection

The design of the artificial neural network depends on its connection, activation function, and the training method. So, there are many similarities between the various structures of neural network algorithms. The plurality of the differences appears from the different learning methods and how those methods can adapt the topology of the network.

This section covers a number of artificial neural networks which are the most widely used. They are arranged in several categories of applications. The categories are listed in table below are not meant to be limited, they just want to eliminate some of the ambiguity among different neural network structures and they are suitable to particular applications (Anderson and McNeill, 1992). In general, most of the neural network applications belong to the following five groups as demonstrated in Table (3.1) below:

Table 3.1. Network Selector.



Issue type	Usable networks	Description
prediction	• Back- propagation • Delta bar Delta • Extended delta bar delta • Directed random search • Higher order neural networks • Self-organizing map into back- propagation	Use input values to predict some output (e.g. pick the best stocks in the stock market, predict the weather, identify people with cancer risks)
classification	• Learning vector quantization • Counter propagation • Probabilistic neural network	Use input values to determine the classification (e.g. is the input the letter A. is the blob of video data a plane and what kind of plane is it)
Data association	• Hopfield • Boltzmann machine • Hamming network • Bidirectional associative memory • Spatio-temporal pattern recognition	Like classification but it also recognizes data that contains errors (e.g. not only identify the characters that were scanned but also identify when the scanner wasn't working properly)
Data conceptualization	• Adaptive resonance network • Self-organizing map	Analyze the inputs so that grouping relationships can be inferred (e.g. extract from a data base the names of those most likely to buy a particular product)
Data filtering	• Recirculation	Smooth an input signal (e.g. take the noise out a telephone signal)

3.8. Neural Networks Applications

In this section, we will present some of the fields where artificial neural networks are being applied (Dreyfus, 2005; Engineering, 2018).

- **Language Processing:** This field of applications comprises the conversion of text-to-speech, language detection and translation, secure voice keyed locks, automatic reproduction, the ability to interact with voice orders and processing of natural language.

- **Character Recognition**: Neural network has the ability to distinguish hand printed characters by a scanner device. Recognizing characters and cursive can be done by a special software tool (Quantum Neural Network package).

- **Image Compression:** Neural networks algorithm can be designed and implemented for real-time data compression and decompression applications. These artificial algorithms can minimize an 8- bits of data to 3- bits and then reverse this operation to construct 8-bits again from 3-bits.
- **Signal Processing:** Some of the neural network algorithms have the capability of eliminating electronic noise. Moreover, some applications based on neural networks can find out the engine misfire depend on the engine sound.

- **Banking & Finance:** Neural network algorithms are applied successfully to some banking & finance applications such as future price forecasting, forecasting of the exchange rate, and stock performance. These days, artificial neural networks are usually utilized for financial decision making.

- **Predictions:** Producing the output which is expected can be done by neural networks after giving it a particular input. If we have a neural network which is trained well in a stream of values, then, we can utilize it to anticipate future results. Prediction of Weather conditions (Temperature, humidity, pressure, etc.) is an example of neural network prediction.

- **Medicine:** Studies are still ongoing in the field of using neural networks in medicine. In this time, a lot of research is done on modeling human body parts and distinguishing disease from different scans.

3.9. Introduction to Fuzzy Logic

A Fuzzy Logic System (FLS) is a kind of nonlinear mapping system which is used to set input data to a scalar output data. In general, the architecture of any FLS can be divided into three parts: Fuzzification, Inference system and Rules, and the last part is Defuzzification (G. Chen and Pham, 2000; Sivanandam et al., 2007). These components are shown in Figure 3.11.

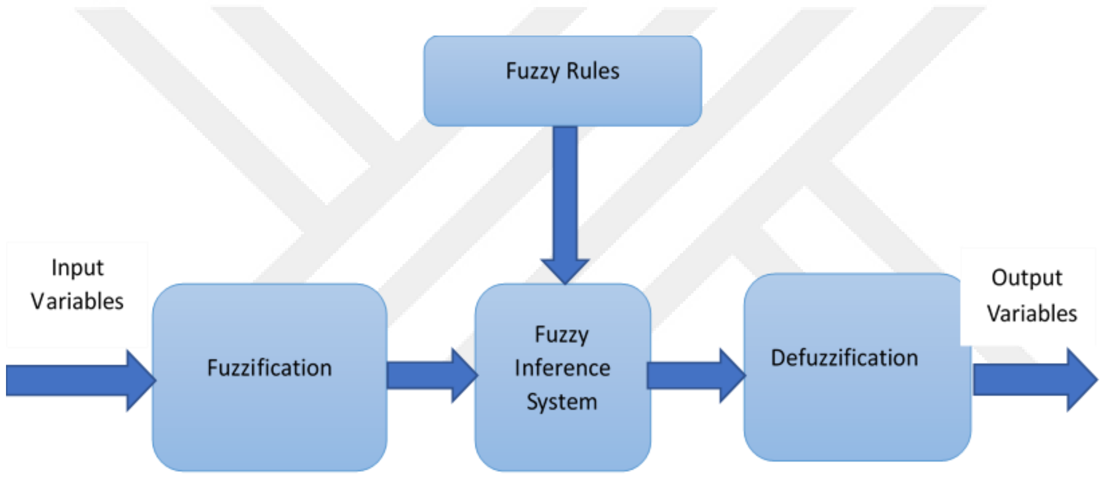


Figure 3.13. Architecture of Fuzzy Logic System.

The Fuzzy logic term is an expansion of Boolean logic which is introduced by Lotfi Zadeh in 1965 depending on the mathematical concept of fuzzy sets, that is a popularization of the traditional set theory (Sivanandam et al., 2007). The FLS has been utilized for many domains, such as control systems and artificial intelligence. Fuzzy logic system is a form of multi-value logic. So, the truth values of any variable could be a real value between zero and one.

It is utilized to deal with the connotation of partial truth, that means, the truth value of variables ranges between totally true and totally false. Compared with traditional two-valued Boolean logic, the truth values of any variable could be only the integer value zero or one. Formalizing human reasoning is one benefits of using fuzzy logic because the fuzzy rules are constructed by natural language. For instance, in Table 3.2**,** there are a few rules of behavior which a car's driver should follow when he is approaching the traffic light:

Table 3.2. Car's driver rules.

Light Color	Speed	Distance	Action
If light: red	if speed: high	If traffic light: close	I brake hard.
If light: red	if speed: low	If traffic light: far	I preserve my speed.
If light: orange	if speed: average	If traffic light: far	I brake softly.
If light: green	if speed: low	If traffic light: close	I accelerate my speed.

Based on Table 3.2, the input variables are approximately estimated by the human brain, for example, the level of verification of a condition inside a fuzzy system.

3.9.1. Classic and Fuzzy Sets

The concept of a traditional set is one of the rudimentary notions, that do not have a clear definition. Commonly, a set is known as a group of elements that have some common characteristics distinguishing these elements from other elements, like a group of negative numbers less than -30 or the set of flying fishes.

In general, uppercase letters are used to denote any set (A, B, . . .), while lowercase letters are used to denote objects within any set (an object x, z, . . .).

In the case of traditional set (classic sets), any particular object x could be a member of a set A (it has value=1) which is symbolized by $x \in A$ or not belong to set A and is symbolized by $x \notin A$ (it has value=0).

There are many basic operations that can be applied on classic sets as demonstrated below (Jamshidi and Zilouchian, 2001; Tettamanzi and Tomassini, 2013):

- Intersection or Min function (AND logic)

$$A \cap B = \{x \in and\, x \in B\}$$

Or:

$$min\ (\mu A(x), \mu B(x)) \qquad (3.16)$$

- Union or Max function (OR logic)

$$A \cup B = \{x \in A\ or\ x \in B\}$$

Or:

$$max\ (\mu A(x), \mu B(x)) \qquad (3.17)$$

- Complement or Negation (NOT logic)

$$A = \{x \notin A\}$$

Or:

$$1 - \mu A(x) \qquad (3.18)$$

On the other hand, fuzzy logic is depending on the fuzzy sets theory, that is a generalization of the traditional set theory and that means, the traditional set theory is a part of the fuzzy sets theory. Figure 3.12 demonstrates this relationship.

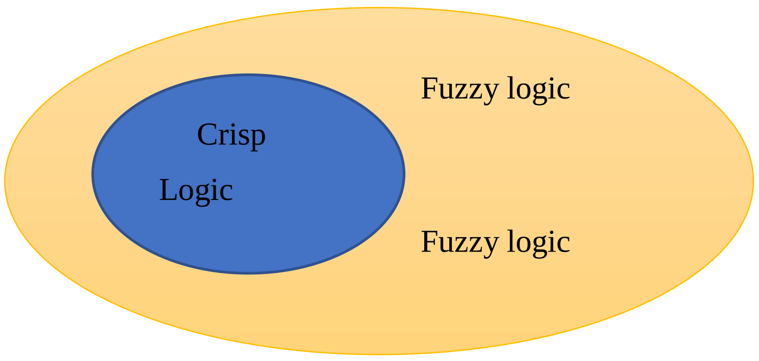


Figure 3.15. Relationship between Fuzzy and Classic Sets.

In a fuzzy logic system, an object x in the fuzzy set A could belong to set A with different membership degrees between 1 (full membership) and 0 (lack of membership). In order to characterize a fuzzy set A, we have to prepare its membership function where:

$$\mu A : X \rightarrow [0, 1].$$

There are different types of membership functions that can be applied for fuzzy sets(G. Chen & Pham, 2000)(Sumathi & Paneerselvam, 2010), three of them are introduced which are used in most common applications as shown below:

- Gaussian membership function: which describe by Eq. 3.18 and Figure 3.13

$$\mu A\ (x, a, b) = exp\left(-\ \frac{(x-a)^2}{2b^2}\right) \tag{3.18}$$

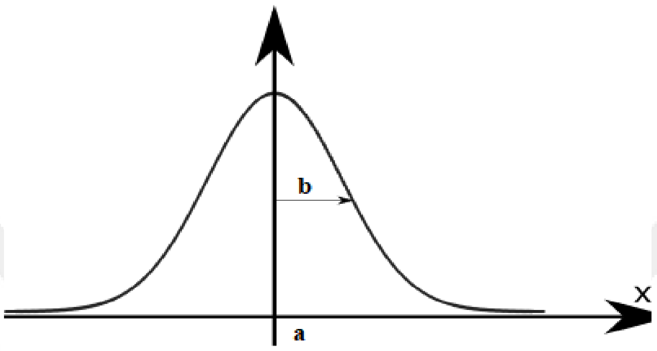


Figure 3.16. standard Gaussian Membership.

Where a and b are its parameters which specify the center of the Gaussian function and control the width of the function curve respectively.

- Trapezoidal membership function: Eq. 3.12 and Figure 3.14 described this function

$$\mu A(x, a, b, c, d) = f(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x \leq b \\ 1 & b < x \leq c \\ \frac{d-x}{d-c} & c < x \leq d \\ 0 & x > d \end{cases} \tag{3.19}$$

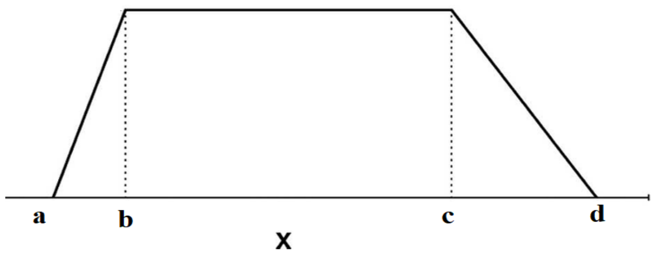


Figure 3.17. General Form of Trapezoidal Membership.

Where: **a=** lower limit, **d**= upper limit, **b** and **c**= lower and upper support limit respectively, where **a < b < c < d**.

- **Triangular function:** this function can be described by Eq. 3.20 and Figure 3.15 where **a** and **b** are lower and upper limit respectively and m is point where **a < m < b**

$$\mu A(x,a,b,m) = f(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{m-a} & a < m \leq b \\ 0 & x > b \end{cases} \tag{3.20}$$

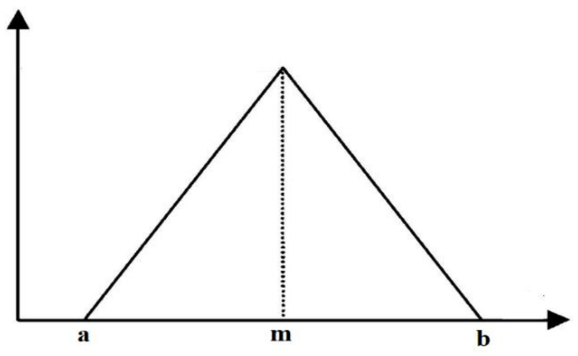


Figure 3.18 General Form of Triangular function.

3.9.2. The linguistic variables

Input and output data for a fuzzy system can be described by linguistic terms whose values are a natural language which include words or sentences, rather than numerical values. Usually, the linguisttic variable can be broken down into a small group of linguistic terms (Sumathi and Paneerselvam, 2010).

Example: Suppose that we have an air conditioner system which illustrates in Figure 3.16**.** Assume that the temperature (t) is the linguistic term that expresses the status of the room temperature. To introduce the temperature in linguistic language, terms such as and **cold, very-cold, warm**, and **hot** are utilized in real life. Those words are called the temperature linguistic values. So, the temperature of linguistic values can be:

T(t) = {very-hot, hot, warm, cold, very-cold, ..}. Each member of this set is known as a linguistic term and can cover a part of the comprehensive temperature values (Figure 3.17).

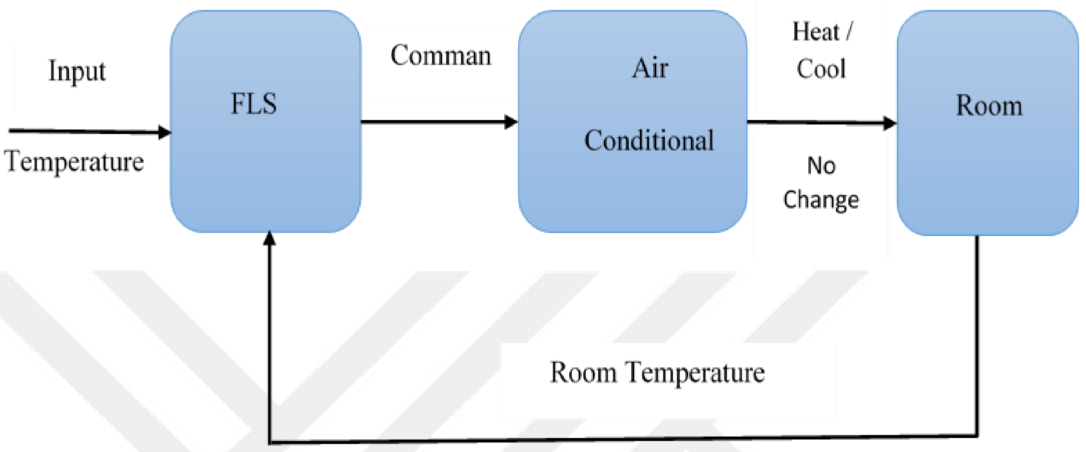


Figure 3.19. Air Conditioner based on Control of Fuzzy Logic System.

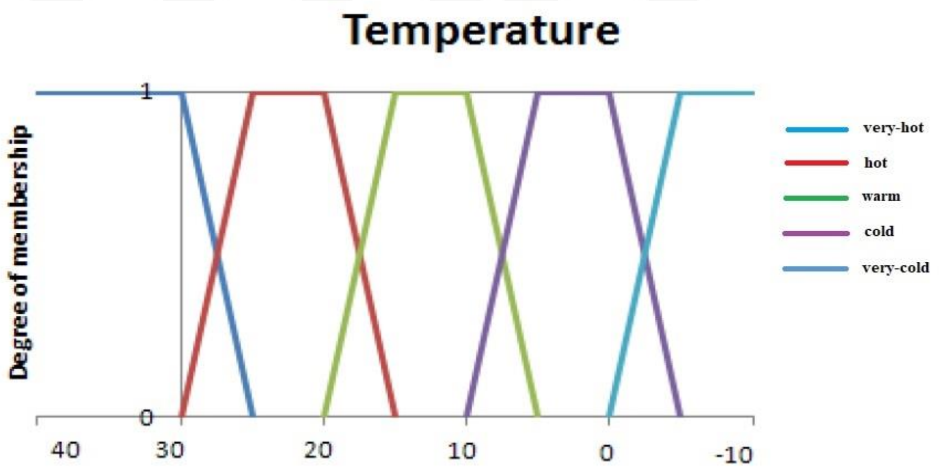


Figure 3.20. The Linguistic Terms of the Temperature

3.9.3. Fuzzy Rules (Reasoning in fuzzy logic)

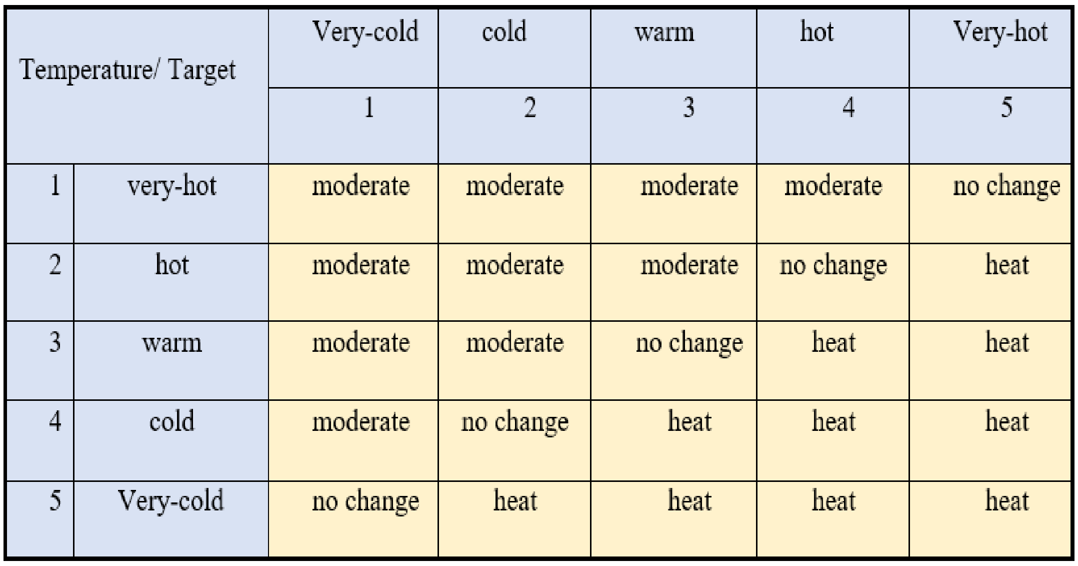
In a fuzzy logic system, the final system output is produced based on the constructed fuzzy rules. Any fuzzy rule can be constructed by a basic IF-THEN condition (Kacprzyk and Pedrycz, 2015; Tettamanzi and Tomassini, 2013). In Table 3.3, an example of fuzzy rules that are built to be used in air conditioner system in Figure 3.16, while the Table 3.4 presents the matrix of all possible fuzzy rules that can be used for the system in Figure 3.16. According to Table 3.4, the Row fields comprise the values of the current room temperature, while column field comprises the target temperature values. The remaining cells are the resulting actions when row and column values are intersected. For example:

Cell (4, 4) in Table 3.4 can be read as: If current temperature is cold **AND** target is hot **THEN** the action will be heat.

Table 3.3. Example of Air Conditioner Fuzzy Rules.

Fuzzy system rules	
1	**If** (temperature = warm) **AND** (target = warm) **THEN**: action = no change
2	**If** (temperature = hot **OR** very-hot) **AND** (target = warm) **THEN**: action = moderate
3	**If** (temperature = cold **OR** very-cold) **AND** (target = warm) **THEN**: action is heat

Table 3.4 Example of Fuzzy Rules Matrix.



Temperature/ Target		Very-cold	cold	warm	hot	Very-hot
		1	2	3	4	5
1	very-hot	moderate	moderate	moderate	moderate	no change
2	hot	moderate	moderate	moderate	no change	heat
3	warm	moderate	moderate	no change	heat	heat
4	cold	moderate	no change	heat	heat	heat
5	Very-cold	no change	heat	heat	heat	heat

3.9.4. The Defuzzification

The final result of the FLS is a fuzzy value after fuzzy rules are applied within the inference engine. So, this result must be converted to have a final crisp output after performing the defuzzification step. The defuzzification step is applied based on different kind of membership functions. Here, we will introduce some of the most common membership functions which are used in the defuzzification stage. Some numerical examples related with this defuzzification types can be found in (Sivanandam et al., 2007; Irwin, Wilamowski, and Irwin, 2011).

- **Center of Sums technique (COS):** In this technique, the overlapping area is calculated two times. The defuzzified value **x** is expressed as Eq. 3.21:

$$x = \frac{\sum_{i=1}^{N} x_i * \sum_{k=1}^{n} \mu A_k\,(x_i)}{\sum_{i=1}^{N} * \sum_{k=1}^{n} \mu A_k\,(x_i)} \qquad (3.21)$$

Where: **n** is the fuzzy sets number, **k** the number of the fuzzy sets, **N** is the counter of fuzzy input data, and $\boldsymbol{\mu A_k}\ (\boldsymbol{x_i})$ represent the membership function.

- **Centroid of Area (COA) technique:** This technique supplies a crisp output by calculating the center position of gravity of the fuzzy set. The entire distribution area of the membership function that utilized to perform the connected control is split into some sub-areas. Then. the summation of the centroid of these sub-areas is performed in order to get the final defuzzified value as show in Eq. 3.22.

$$x = \frac{\sum_{i=1}^{N} x_i * \mu\,(x_i)}{\sum_{k=1}^{n} \mu(x_i)} \qquad (3.22)$$

Here, **n** is the number sample, $\boldsymbol{\mu(x_i)}$ represents the membership function, $\boldsymbol{x_i}$ indicates the input sample element.

- **Weighted Average technique:** This method does not need a high level of mathematical computation. This technique is only used for symmetrical membership functions and the produced outputs are very similar to the centroid of area technique. The weighted of each membership function is done by taking its maximum value. This method can be expressed as shown in Eq. 3.23:

$$x = \frac{\sum \mu\,(x_i)\,.\,x_i}{\sum \mu\,(x_i)} \qquad (3.23)$$

- **Mean of Maxima technique (MOM):** Also known as middle-of-maxima. In this technique, the defuzzified output is calculated based on the highest value of the membership. If there more than one sample having maximum values of the membership, then the average value of all these values are taken. Simply, Eq. 3.24 and Figure 3.18.

$$x = \frac{a+b}{2} \qquad (3.24)$$

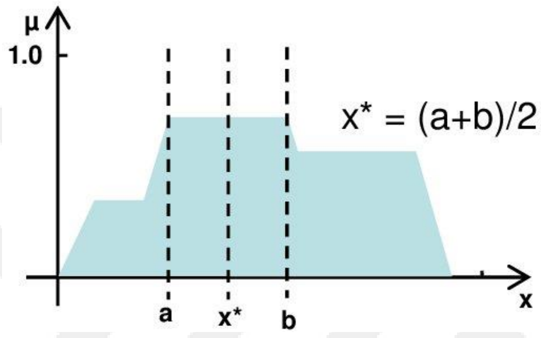


Figure 3.21. Mean of Maxima Defuzzification Method.

3.10 Advantages and Disadvantages of Fuzzy Logic system

Advantages:

- A simple and straightforward intelligent approach.
- Easy to realize and achieve.
- Provide a simple method of presentation.
- Provide an efficient performance.

Disadvantages:

- Model development is complicated from a fuzzy system.
- The system needs a high smooth tuning and simulation in order to implement.
- Hard to select an appropriate membership function to be used in fuzzy systems.
- Knowledge about the solution is required to find the problem solution. This
- knowledge must be ready in if-then form.

3.11. Introduction to Neuro-Fuzzy System

Having a good performance with fuzzy system development is a complicated task. The process of finding membership functions and suitable rule sets are sometimes an exhausting task based on attempt and error. So, the concept of integrating artificial learning algorithms with fuzzy systems has appeared. The artificial neural networks that have powerful learning techniques have been introduced to assist in the fuzzy system's tuning and development(Kaynak, Zadeh, Turksen, and Rudas, 1996).

These fuzzy logic systems and neural networks techniques are usually utilized in order to deal with problems when the traditional methods do not provide a simple and precise solution (P. Liu and Li, 2004).

In general, the neuro-fuzzy term can be defined as a kind of system which is characterized by a similar structure of a fuzzy logic system where the neural networks algorithms are utilized to adjust reasoning rules and fuzzy sets in an iterative form based on data system (input and output vectors). This kind of systems illustrates different behavior within two levels. During the first level, which is known as a learning phase, it behaves similar to any neural network that training its internal parameters iteratively. While in the second level or execution phase, the system behaves similar to a classical fuzzy logic system. The integration between these two techniques provide a best performance when compared the systems results with each isolated technique (Fullér, 2013; Siddique and Adeli, 2013).

3.11.1. Types of Neuro-Fuzzy Systems

There are several combinations between neural network and fuzzy system techniques that can be split, into the following criteria (Abraham, 2001; Vieira, Dias, and Mota, 2004; Nikam, Nikumbh, and Kulkarni, 2012):

- **Cooperative Neuro-Fuzzy System:** In this type of cooperative systems, the neural networks approach acts as a pre-processing unit where the selected algorithm of neural networks is used to train some parameters of the fuzzy system such as fuzzy sets, parameters of membership functions, and fuzzy rules. After this phase, the learning method of neural networks are taken away and the fuzzy system works independently. Figure 3.19 shows this concept.

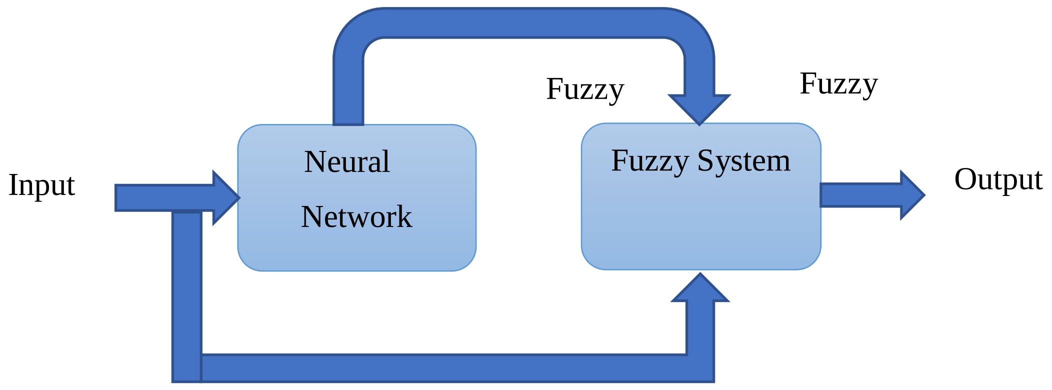


Figure 3.22. Cooperative Neuro-Fuzzy.

- **Concurrent Neuro-Fuzzy System:**

In the concurrent approach, the fuzzy system and the neural network do their tasks together continuously. That means the input vector is first pre-processed by the fuzzy system, and then the learning algorithm of neural network handles the results of the concurrent system. However, the system's outputs are not completely interpretable, and this can be considered as one of the system's disadvantages. Figure 3.20 shows this idea.

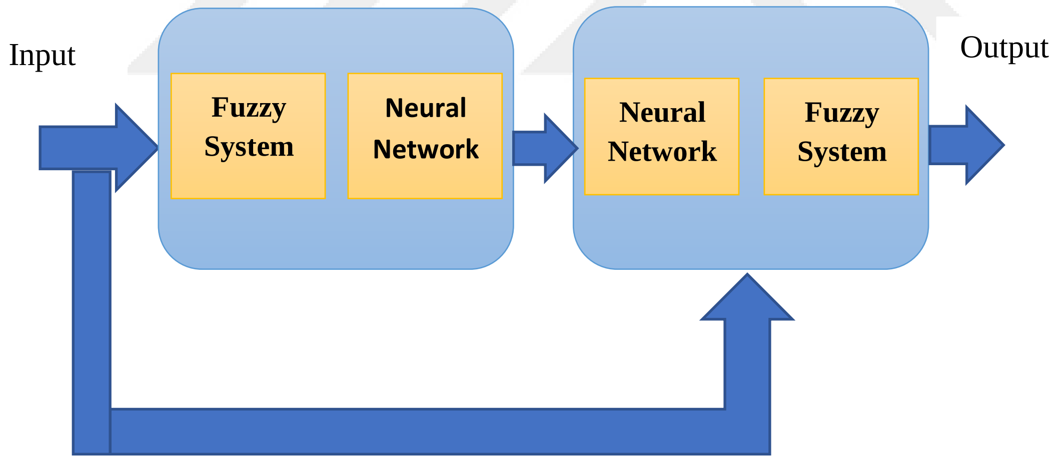


Figure 3.23. Concurrent Neuro-Fuzzy System

- **Hybrid Neuro-Fuzzy Systems**

In this system, an artificial learning approach based on the gradient algorithm is integrated with a fuzzy system to generate a hybrid neuro-fuzzy system. The gradient algorithm (will be explained later) is utilized to its parameters: fuzzy rules and its weights, membership functions, and fuzzy sets) over input and output patterns. The simplicity of the system functionality is one of the system advantages. Also, the representation of the data over a neural network within the hybrid system is more suitable because it permits to visualize the flow of data and error signals over the system in order to modify its parameters. There are different architectures which are used to construct hybrid neuro-fuzzy systems (Vieira et al., 2004). Some of the hybrid system models are expressed by artificial neural networks which perform logical functions. Adaptive Neuro-Fuzzy Inference System (ANFIS) is an example which is the most used architecture of a hybrid system.

3.11.2. Adaptive Neuro-Fuzzy Inference System (ANFIS)

In 1993 Jang, J.S.R. has proposed a type of hybrid artificial intelligent system which has some characteristics between neural network and fuzzy system that is ANFIS algorithm. The objective of ANFIS algorithm is to detect a mapping or relationship between the presented inputs data and the system target (desired output) (Jang, 1993). As we mentioned before, there is no ability to learn in the fuzzy logic system. So, the system is unable to modify its parameters.

On the contrary, ANFIS algorithm performs the learning capabilities of neural network to adapt the membership functions in fuzzy logic system automatically(Kaynak, Zadeh, Türksen, and Rudas, 2012). Unlike other traditional FIS, ANFIS algorithm does not need any expert knowledge to realize input-output vectors for fuzzy training. This facility is very important to minimize the complexity of calculating fuzzy training data task.

Another feature of ANFIS algorithm is that extensive initialization of its parameters is not necessary because of a simple random initialization of training can be used. Like any fuzzy logic system, ANFIS algorithm has some architecture models with various techniques for adapting parameters of membership function based on NNs during training phase.

In this thesis, the Takagi–Sugeno inference structure that has a high computational efficiency was utilized to construct the ANFIS model with adaptive capability. This benefit allows the inference algorithm to work efficiently with automatic adaptive mechanism likes ANNs. Since ANFIS parameters are adapted through the training phase, various membership function forms are provided continuously until the training is finished. In general, the Takagi–Sugeno structure is constructed from five layers. Each layer comprises several nodes with different tasks (Aliev and Guirimov, 2014) as illustrated in Figure 3.21. The fuzzy rules in Takagi–Sugeno model are produced from a set of input-output pairs. In this model, the rule can be constructed as shown below:

$$\boldsymbol{if}\ x_1\ is\ A_1\ and\ x_2\ is\ A_2, \dots, AND\ x_m\ is\ A_m;\ \boldsymbol{then}\ O = f(x_1, x_2, \dots, x_m);$$

Where:

x_1,…, *x_m* are the input data; *A_1*,…, A_m are fuzzy sets; and *O* is the output function which can be constant or linear based on the complexity of the application.

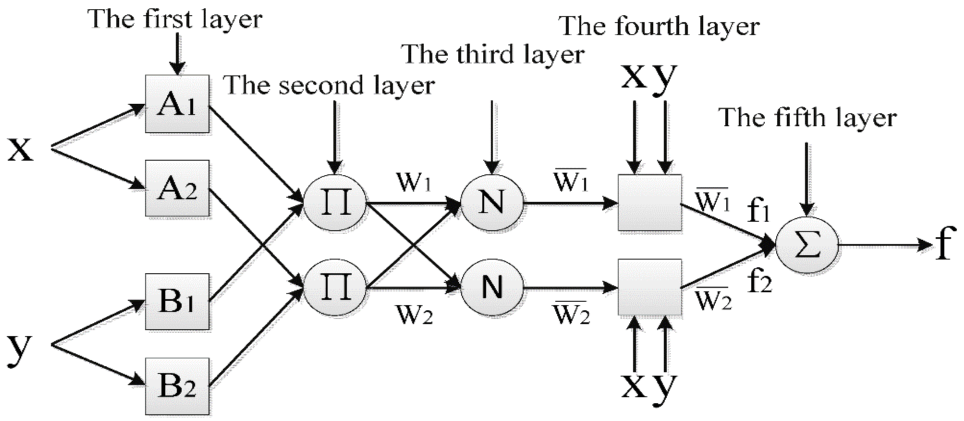


Figure 3.24. General Structure of ANFIS base on Takagi–Sugeno

3.11.2.1. ANFIS Architecture

For simplicity and based on Figure 3.21**,** suppose that we have inputs **x** and **y**, and **f** as a linear output. Two fuzzy rules are used in the form of "If-Then" based on Takagi–Sugeno model(Jang, 1993), as illustrated:

$$Rule\ 1 = \boldsymbol{If}\ x\ is\ A_1\ and\ y\ is\ B_1\ \boldsymbol{Then}\ \ f_1\ = p_1\ x + q_1\ y + r_1$$

$$Rule\ 2 = \boldsymbol{If}\ x\ is\ A_2\ and\ y\ is\ B_2\ \ \boldsymbol{Then}\ \ f_2\ = p_2\ y + q_2\ y + r_2$$

where A1, B1 and A2 , B2 are representing the membership functions which are associated with each input x and y. while $f_1\ and\ f_1$ are a first order linear function which have p_1, q_1, r_1 and p_2, q_2, r_2 as linear parameters. So, a specification of each layer is described as shown below:

- **Layer 1:** In the first layer, each node in this layer adjusts to a function parameter. So, each node's output is a membership degree of linguistic terms that is generated by the membership functions inputs. Gaussian membership function in Eq. 3.25 is a membership example which can be utilized within the first layer:

$$\mu_{Ai}(x) = \frac{1}{1+\left|\frac{x-c}{a}\right|^{2bi}} \quad , \ i = 1,2,3, \qquad (3.25)$$

$$Q_{1,i} = \mu_{Ai}(x), \ i = 1,2 \qquad (3.26)$$

$$Q_{2,i} = \mu_{Bi}(y), \ i = 3,4 \qquad (3.27)$$

Where $\mu_{Ai}(x)$ and $\mu_{Bi}(y)$ are membership functions degree for both A_i and B_i (Figure 3.21) fuzzy sets respectively. While (a, b, and c} are the parameters of Gaussian membership function that are used to adapt the membership function shape. All parameters in this layer are referred to premise parameters.

- **Layer 2:** All nodes in this layer are non-adjustable or fixed nodes and each node is denoted as **Π**. The nodes output represents the firing strength of each constructed rule based on "min" or "prod" operator. Usually, Logical "AND" is commonly utilized in this layer as shown in Eq. 3.28

$$Q_{2i} = w_i = \mu_{Ai}(x) * \mu_{Bi}(y), \ i = 1,2$$

(3.28)

- **Layer 3:** Like layer 2, all layer nodes are non-adjustable or fixed nodes and the nodes are denoted by **N** label. Each node is a determining the ratio between each firing strength of the rule and the aggregation of all rules' firing strengths. This result is called a "normalized firing strength" which can be expressed by Eq. 3.29.

$$Q_{3,i} = \bar{w}_i = \frac{w_i}{w_1+w_2}, \quad i = 1,2,3, \qquad (3.29)$$

where $\bar{w}_i$ represents the output of firing strength.

- **Layer 4:** This layer is a defuzzification layer. Each node in this layer is an adaptive node. The product of normalized firing strength of all rule sets that are generated in layer 3 are done in this layer. The nodes calculate the parameters of linear function (*p, q,* and *r)* as illustrated in Eq. 3.30. The Parameters in this layer are known as consequent parameters.

$$Q_{4,i} = \overline{w} f_i = \overline{w}(p_i x + q_i y + r_i\ ),\ i = 1,2,3, \qquad (3.30)$$

Where “p”, “q” and “r” are adaptive consequent parameters.

- **Layer 5:** This layer consists of single non-adaptive node that performs the summation task of all incoming data from layer 4 as shown in Eq. 3.31. The node in this layer is labeled as **Σ**

$$\mathrm{Q_5} = \sum \overline{\mathrm{w}}\mathrm{f_i} = \frac{\sum_i w_i f_i}{\sum_i w_i},\ \mathrm{i} = 1,2,3, \qquad (3.31)$$

3.11.2.2. ANFIS Learning Algorithm

Neuro-adaptive learning method combines with a fuzzy logic system and ANN algorithms to training the information that is extracted from provided data sets. It is utilized to figure the parameters of membership function that enable the related inference engine in a fuzzy system to follow the provided input/output data set. All parameters related to membership functions are modified during the learning phase (Jang, 1993; Kaynak et al., 1996).

To more effectively cooperate with real-world applications, the process of integrating the learning algorithm with ANFIS architecture aims to regulate all adjustable parameters and re-formulate the output of ANFIS in order to match the providing training data sets. To get a better convergence rate, the hybrid ANFIS algorithm can be trained by hybrid training techniques algorithms which comprising gradient descent and least square algorithms can be utilized.

The linear least squares algorithm (LSM) is used in the forward pass to fetch the optimal consequent parameter values on the layer 4, while gradient descent (GDM) based on the back-propagation algorithm is provided in the backward pass to estimate the method of modeling the input/output data by the fuzzy inference. The moment that the optimal values of consequent parameters are located, the backward pass begins. During the backward pass, the calculated error is propagated inversely to the previous layers and the premise parameters in layer 1 are modified by gradient descent method (Jang, 1993; Aliev and Guirimov, 2014). Two passes of the ANFIS hybrid algorithm are shown in Table 3.5

Table 3.5. Hybrid Learning Algorithm within ANFIS Algorithm.

	Forward pass	Backward pass
Premise Parameters	Fixed	Gradient Descent
Consequent Parameters	Least Squares Estimation	Fixed

CHAPTER 4
ANFIS MODIFICATIONS

4.1. Introduction

In chapter 3, we presented a structure and hybrid learning technique background for the Adaptive Neuro-Fuzzy Inference System (ANFIS) algorithm. In this chapter, we will introduce two new modifications for ANFIS algorithm. The gradient descent optimization algorithm known as Momentum factor algorithm is combined with the learning algorithm of ANFIS. This optimization algorithm works to avoid stuck in local minima and accelerate ANFIS algorithm to find the optimal solution. Whereas the extending architecture of the standard algorithm is done in the second modification level. This extending is achieved by adding a new compression layer in order to optimize the adjustable parameters in the output layer. But, we cannot apply these two modifications without writing our ANFIS algorithm that can accept the modifications and compare its performance with the performance of Matlab toolbox.

4.2. Effect of Momentum on ANFIS algorithm

The optimization algorithms in neural networks like gradient descent optimization algorithm are utilized to arrive a global minima by minimizing the error function. The minimizing error function in the idle situation could be like the ball gradient in Figure 4.1.

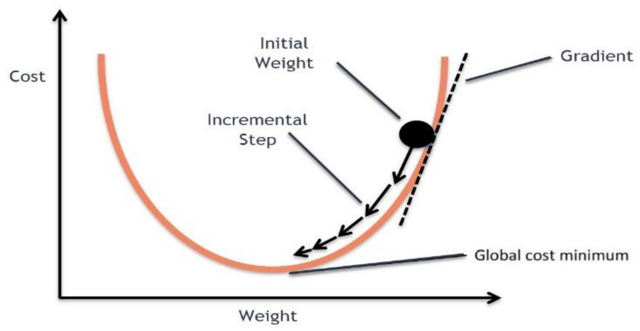


Figure 4.1. Gradient Descent Optimization .

Thus, we are ensured to reach the optimal solution (global optimum) because of no local minimum on the error surface where the optimization can get stuck. But in the real-world, the surface of error is more complicated, it can be composed of many local minima. Figure 4.2 is an example of such an error surface.

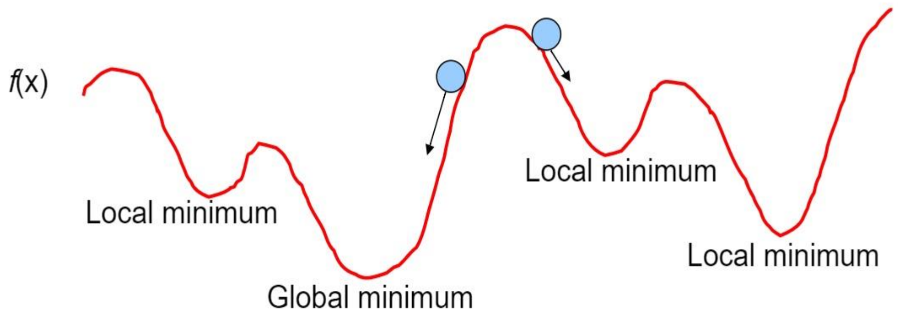


Figure 4.2. Global and Local Minimum.

According to the Figure 4.2, the solution of the algorithm can easily fall and stuck in some local minima (sub-optimal solution). In that time, we think that the algorithm gets the global minima (final optimal solution). The momentum factor which has a value between 0 and 1 plays a key role to avoid this situation. The momentum factor tries to boost the steps size that are taken towards the global minima by jumping over a local minima.

In our research and during the first modification, the momentum factor is merged ANFIS algorithm through a backward learning pass to minimize the error and speed up the training process. To associate a momentum factor to ANFIS algorithm, a fraction “m” that represents a small value of momentum must be added to the equations that are used to update weights and biases. To do that, we have to write our handwritten ANFIS algorithm because the ANFIS toolbox functions in Matlab environment do not support adding momentum factor and there is no way to do that. The handwritten ANFIS algorithm helps the researcher to enhance the ANFIS functionality and tracking it easily without relying on Matlab ANFIS-command. Figure 4.3 shows the pseudo code of ANFIS algorithm and how can associate the momentum factor to the ANFIS algorithm.

```
Set the type of membership function (Gaussian function)
Load initial premise parameters (a,b,c)
Load initial consequent parameters (p,q,r)
Load initial momentum factor (m)
Input the sensor data
Normalize the data


// start forward path //
While (error > threshold) then
    Generate the membership degrees for input
    Normalize and aggregate generated data
    Calculate the output of the algorithm
    Update consequent parameters
    $p_{new}=p_{old}$ + delta(error) * input data
    $q_{new}=q_{old}$ + delta(error) * input data
    $r_{new}=r_{old}$ + delta(error)

// start backward path //
// update premise parameters
    $a_{new}=a_{old}$ + delta(error) + momentum factor (m)
    $b_{new}=b_{old}$ + delta(error) + momentum factor (m)
    $c_{new}=c_{old}$ + delta(error) + momentum factor (m)

End while loop
```

Figure 4.3. Pseudo Code of the Proposed Algorithm.

4.3. Performance of Modified Handwritten ANFIS Algorithm

In order to test the performance of our handwritten algorithm, three different types of data sets are used. First of all, we will compare our algorithm with the ANFIS function in Matlab toolbox. The optimal solutions of discrete and continuous algorithms and applications and perform the data analysis can be achieved by Matlab toolbox functions. The toolbox allows researchers to handle optimization tasks of any design. So, comparing the performance of our handwritten algorithm with Matlab toolbox is the best way to measure the performance of the algorithm. So, three different types of data sets are used for this reason.

4.3.1. Testing ANFIS performance based on Global Ice Volume data set

The first data set consists of 219 samples for Global Ice Volume over the last 440,000 years. This data set is implemented in both ANFIS toolbox and handwritten algorithms. The data set is divided into two folds comprise 70% of the overall data set which represents 155 samples that are used for training both algorithms. While the other fold is used for the testing phase and has 64 samples which represents 30% of the data set. Figure 4.4 and Figure 4.5 show the results after implementing the Global Ice Volume data set in both training and testing phases based on handwritten algorithm respectively.

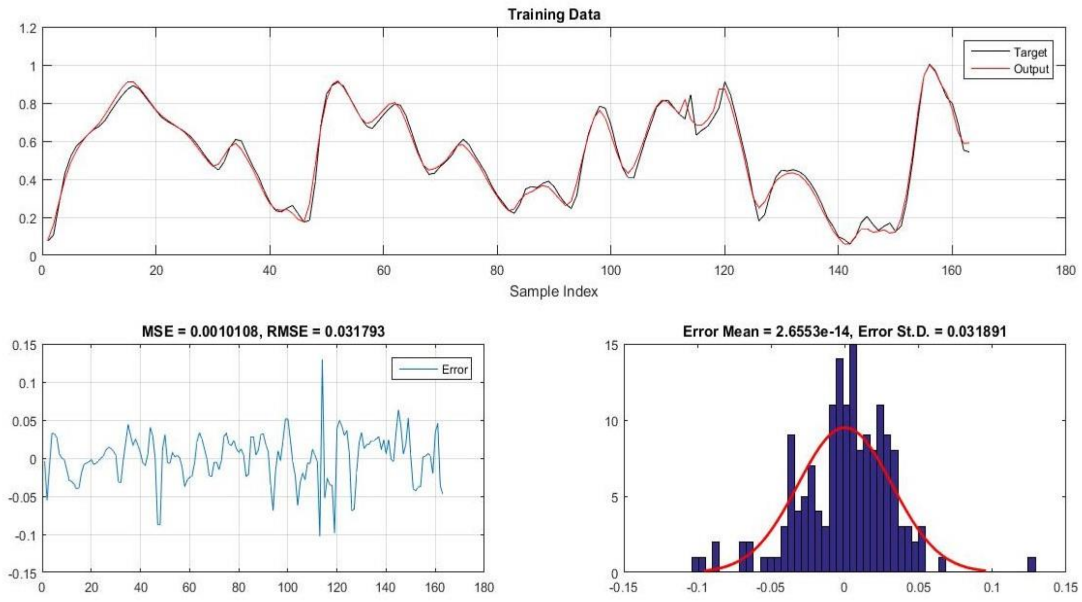


Figure 4.4. Training Algorithm based on Handwritten ANFIS.

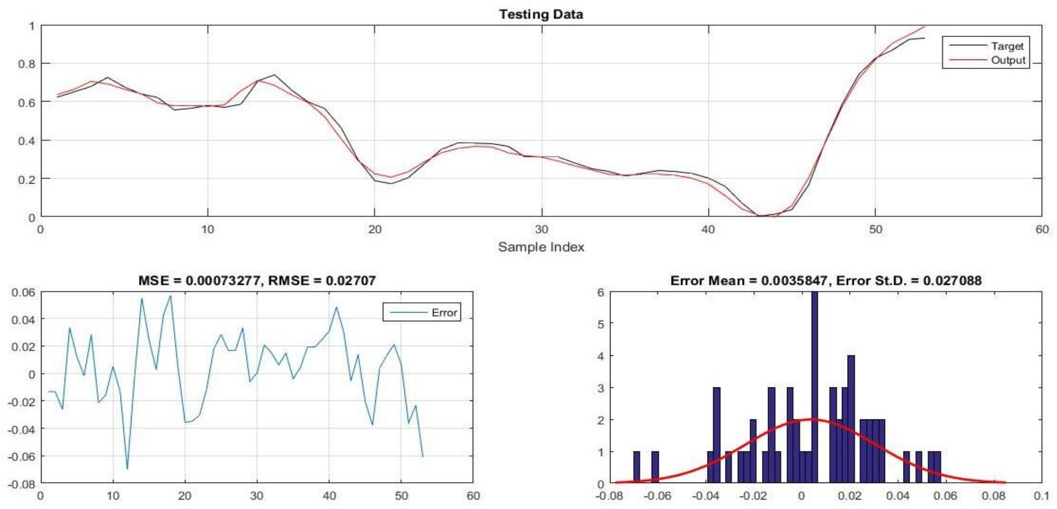


Figure 4.5. Testing algorithm based on Handwritten.

While Figure 4.6 and Figure 4.7 represent the results that are obtained based on ANFIS function in Matlab toolbox.

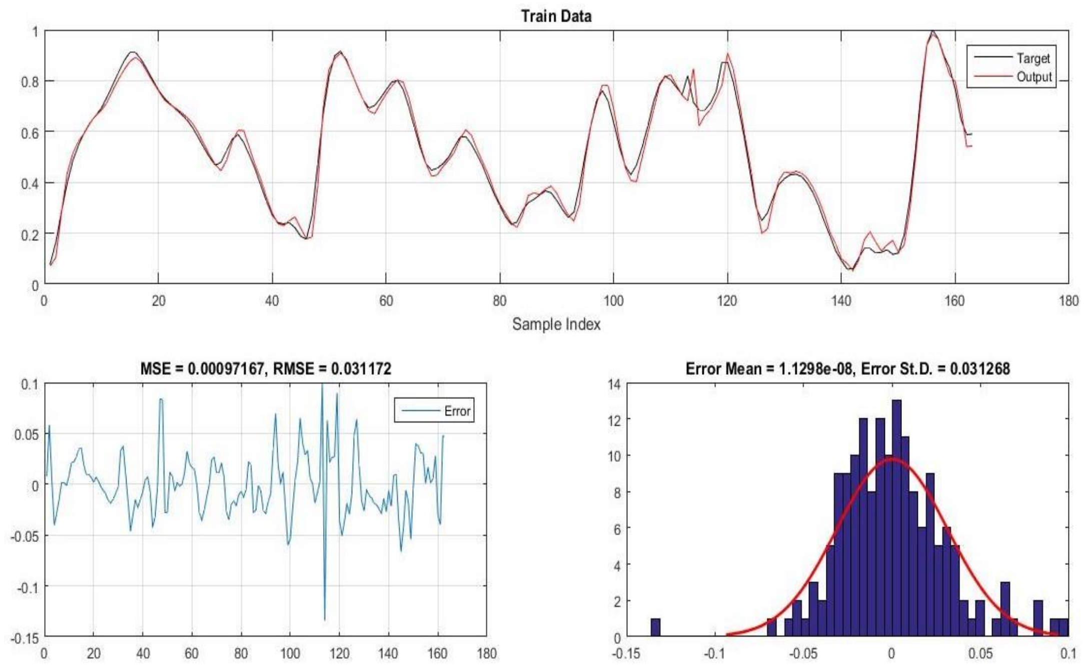


Figure 4.6. Training Algorithm based on Matlab Toolbox.

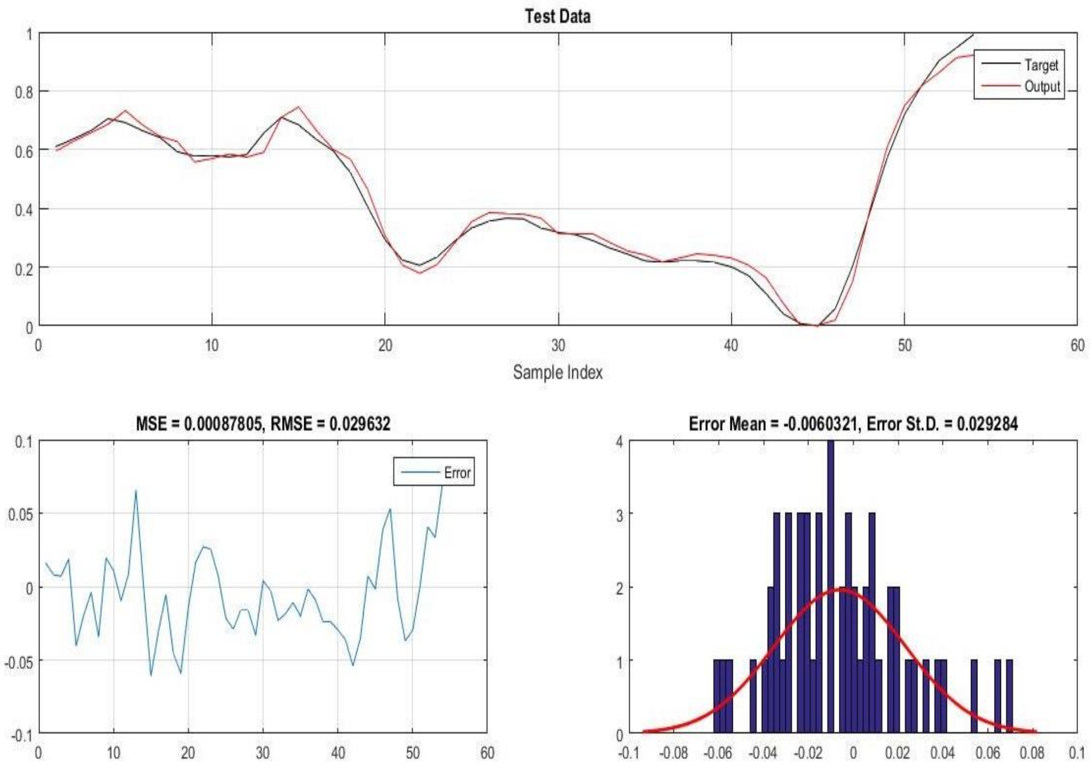


Figure 4.7. Testing Algorithm based on Matlab Toolbox.

Based on above figures that demonstrate the training and testing results of the Global Ice Volume for both Matlab toolbox and handwritten ANFIS algorithms, we can easily see all results are almost the same in both training and testing phases. Matching results in both algorithms mean that our handwritten ANFIS algorithm works correctly and arrives at a high optimization level. Table 4.1 and Table 4.2 summarizes all results in term of MSE (Mean Squared Error), RMSE (Root-Mean-Square Error), Mean Error, and STD (Standard Deviation).

Table 4.1. Training Results of Global Ice Volume.

Technique type	MSE	RMSE	Mean	STD
Matlab Toolbox	0.00097167	0.031172	1.1298e-08	0.031268
Handwritten Algorithm	0.0010108	0.031793	26553e-14	0.031891

Table 4.2. Testing Results of Global Ice Volume.

Technique type	MSE	RMSE	Mean	STD
Matlab Toolbox	0.00087805	0.029632	0.0060321	0.029284
Handwritten Algorithm	0.00073277	0.02707	0.0035847	027088

4.3.2. Testing ANFIS Performance based on Time Series Prediction

Recurrent Time Series Prediction (also known as Time Series forecasters) approach is widely utilized in artificial neural networks especially in feed-forward networks that use a kind of sliding window for the input sequence. Market predictions, weather condition predictions, and network traffic forecasting, stock market forecasting are typical examples of the time series approach. Time series prediction approach is used to predict the potential behaviour of the system in the future depending on current information and the antecedent system state. A Recurrent Neural Network (RNN) depends on the internal state (or internal memory) to handle sequences of inputs over time. Eq. 4.1 shows this relationship.

$$\hat{x}\,(t+1) = f\,(x[t], x[t-1], x[t-2], x[t-T]) \qquad (4.1)$$

Where T is the number of previous inputs (or time delay).

While the Figure 4.8 shows the general structure of the time series approach based on tapped delay line.

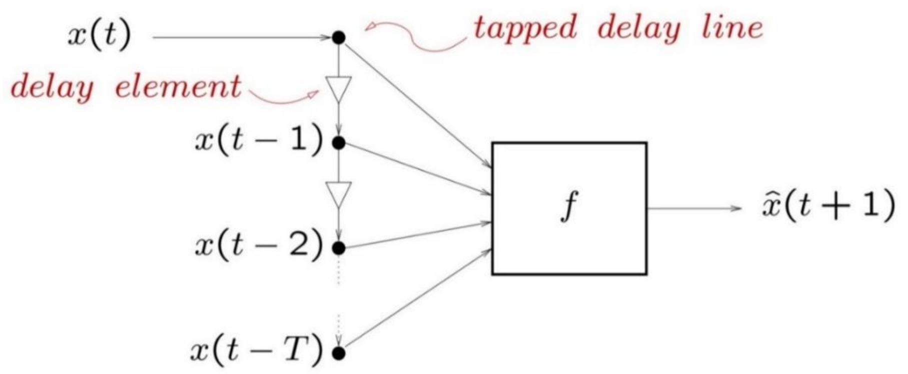
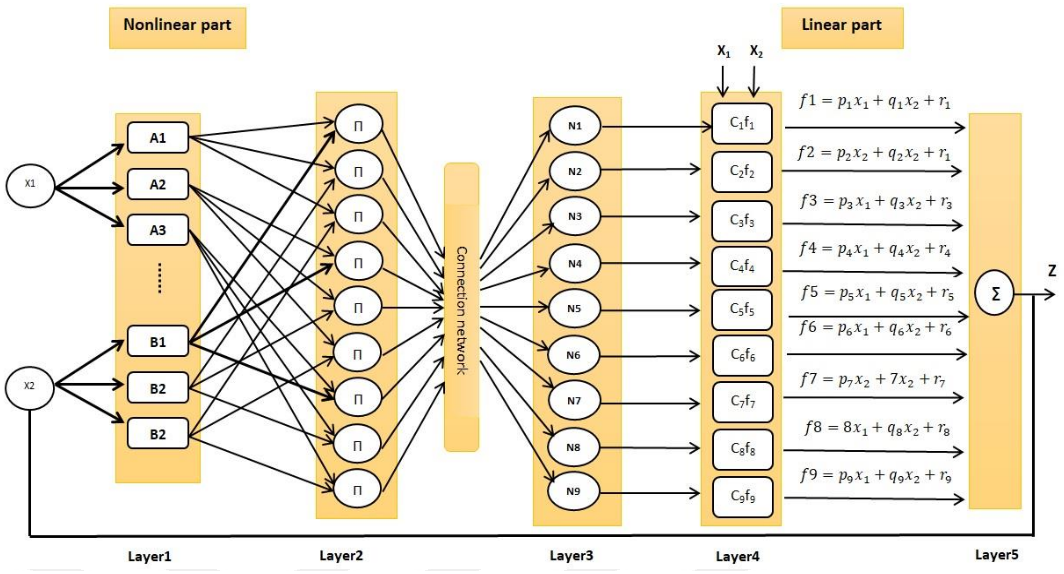


Figure 4.8. A Standard Structure of Time Series Prediction.

Now, our modified algorithm (ANFIS algorithm with momentum factor) can be integrated with the time series approach in order to perform the data prediction of the future behaviour of our hardware system (will be described in the next chapter). Figure

4.9 is an example that shows the structure of ANFIS algorithm with the time series approach (also known as Recurrent ANFIS).



Nonlinear part
Linear part
X1
X2
A1
A2
A3
B1
B2
B2
Π
Connection network
N1
N2
N3
N4
N5
N6
N7
N8
N9
X_1 X_2
C_1f_1
C_2f_2
C_3f_3
C_4f_4
C_5f_5
C_6f_6
C_7f_7
C_8f_8
C_9f_9
$f1 = p_1x_1 + q_1x_2 + r_1$
$f2 = p_2x_2 + q_2x_2 + r_1$
$f3 = p_3x_1 + q_3x_2 + r_3$
$f4 = p_4x_1 + q_4x_2 + r_4$
$f5 = p_5x_1 + q_5x_2 + r_5$
$f6 = p_6x_1 + q_6x_2 + r_6$
$f7 = p_7x_2 + 7x_2 + r_7$
$f8 = 8x_1 + q_8x_2 + r_8$
$f9 = p_9x_1 + q_9x_2 + r_9$
Σ
Z
Layer1
Layer2
Layer3
Layer4
Layer5

Figure 4.9. A General ANFIS Algorithm based on Time Series Prediction.

Two different types of weather data sets that are collected from Chicago park based on beach weather stations (Chicago, 2015b; Chicago, 2015a). These data sets (temperature and humidity) are utilized to measure the performance of our modified algorithm. As we did in the previous section, we will compare all the results with the ANFIS-commands in Matlab Toolbox. Every set of data has 13916 elements that are taken from 5/22/2015 to 12/31/2015.

- **Temperature data set of Chicago park**

Figure 4.10 and Figure 4.11 show training 70% and testing 30% results of temperature data set based on Handwritten Algorithm

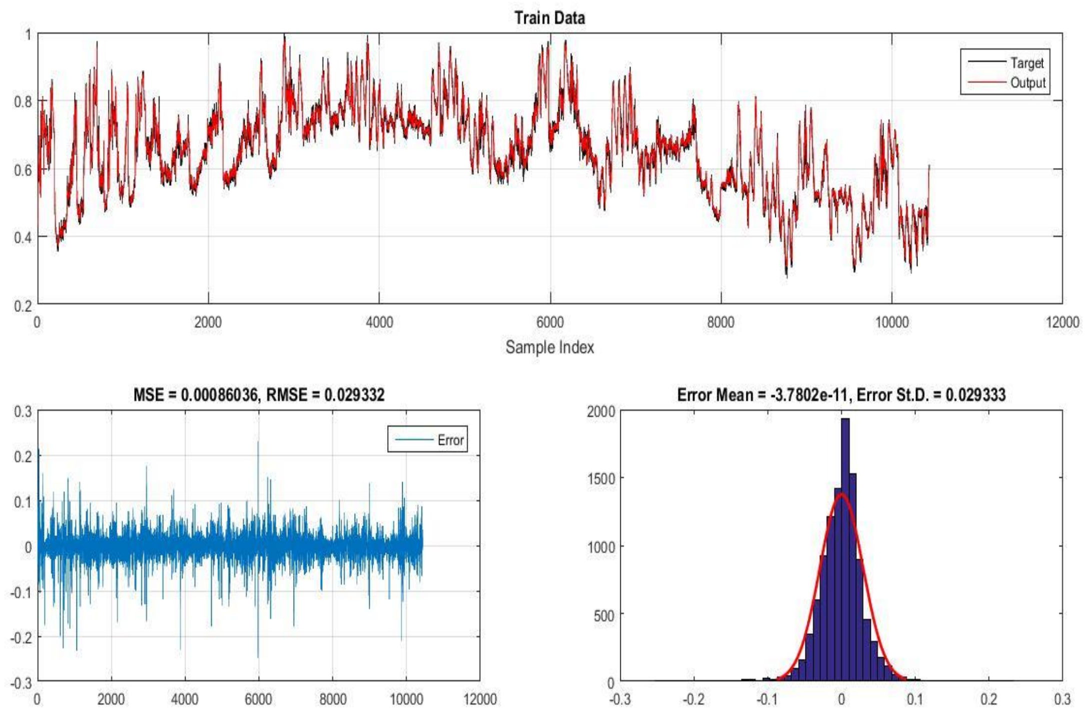


Figure 4.10. Training of Chicago Park Temperature Set Using Handwritten Algorithm.

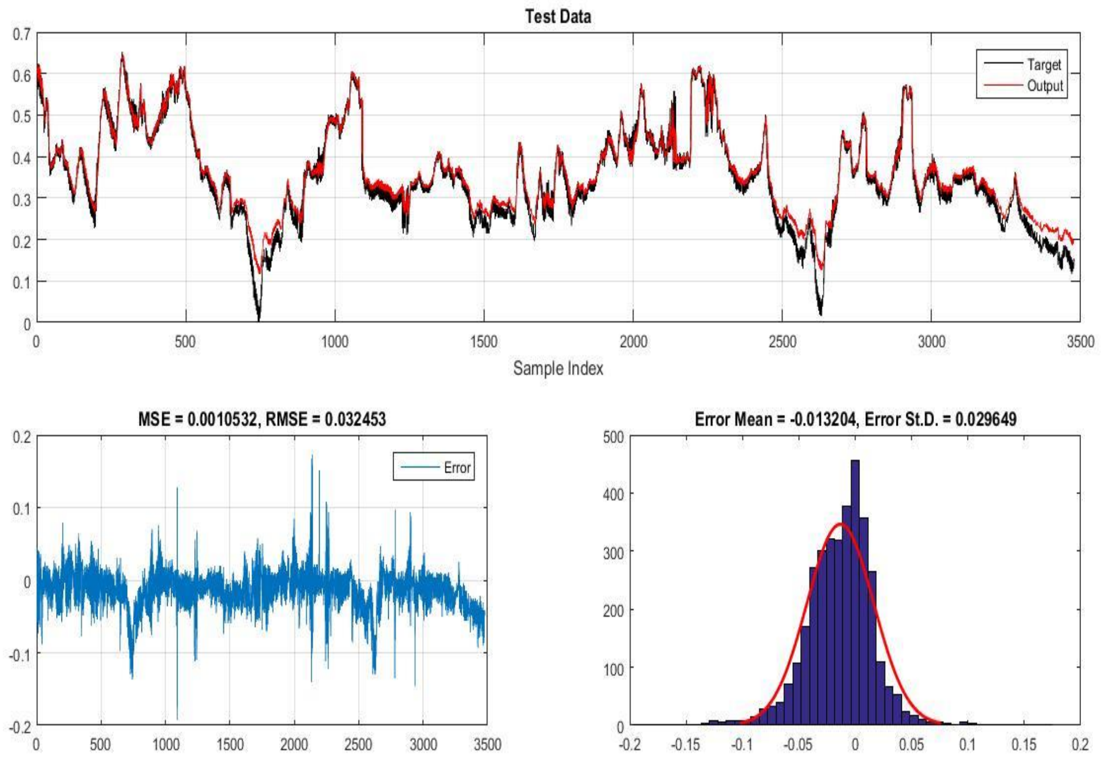


Figure 4.11. Testing of Chicago Park Temperature Set Using Handwritten Algorithm.

While Figures 4.12 and 4.13 show the results (70% training and 30% testing) of the same data set based on ANFIS-commands in Matlab Toolbox.

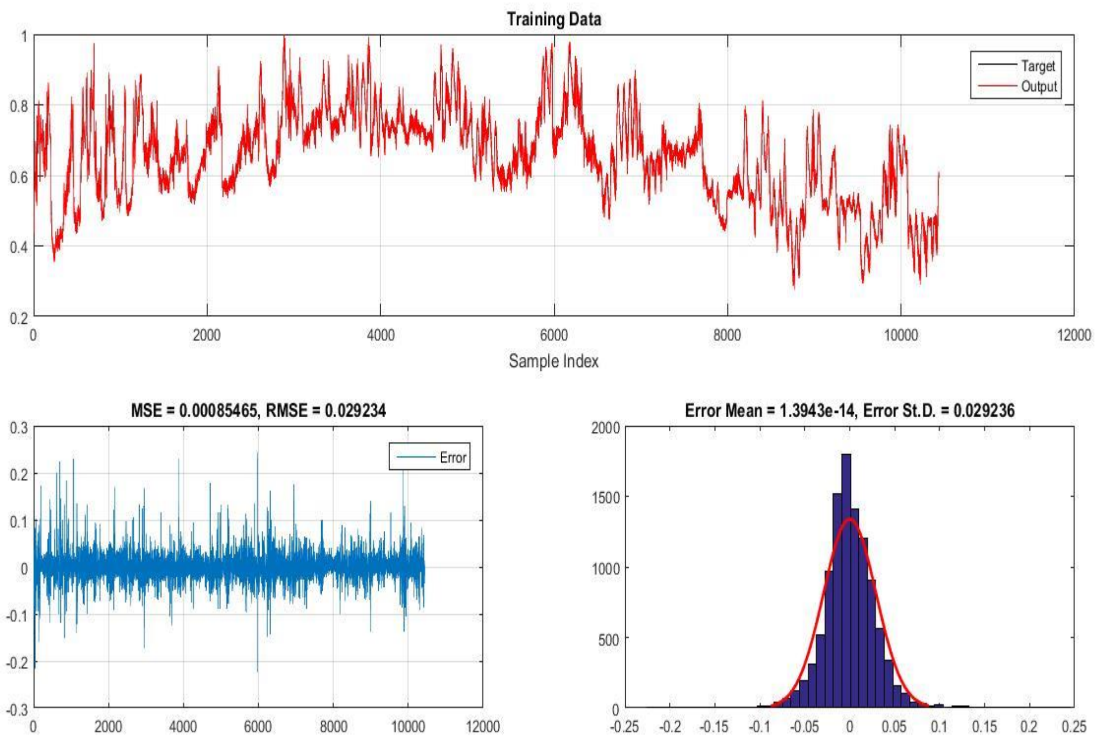


Figure 4.12. Training of Chicago Park Temperature Set using Matlab.

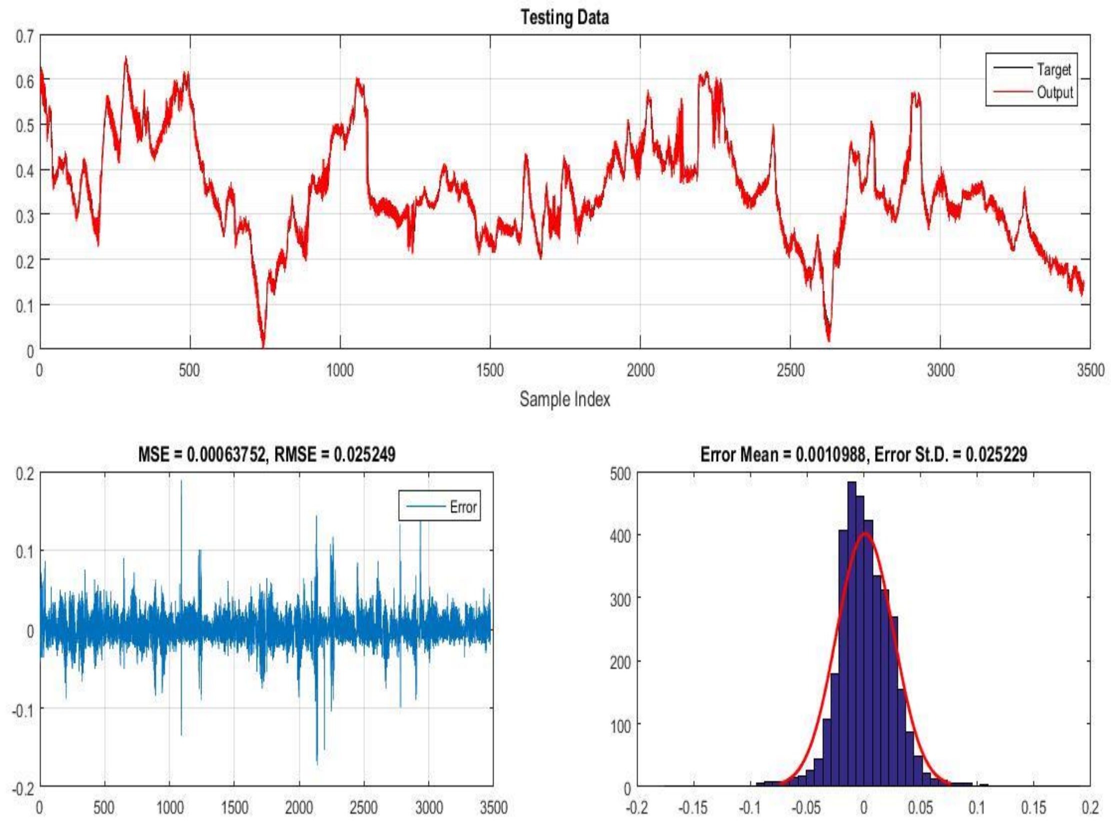


Figure 4.13. Testing of Chicago Park Temperature Set using Matlab.

Table 4.3 summarizes all results (training 70% and testing 30%) in terms of mean error for temperature data set for both algorithms.

Table 4.3. Training and Testing results of Chicago Park Temperature data.

Technique type	Training	Testing
Matlab Toolbox	3.7802e-11	0.013204
Modified Algorithm	1.3943e-14	0.0010988

- **Humidity data set of Chicago park**

Figure 4.14 and Figure 4.15 show training (75%) and testing (25%) results of temperature data set based on Handwritten Algorithm.

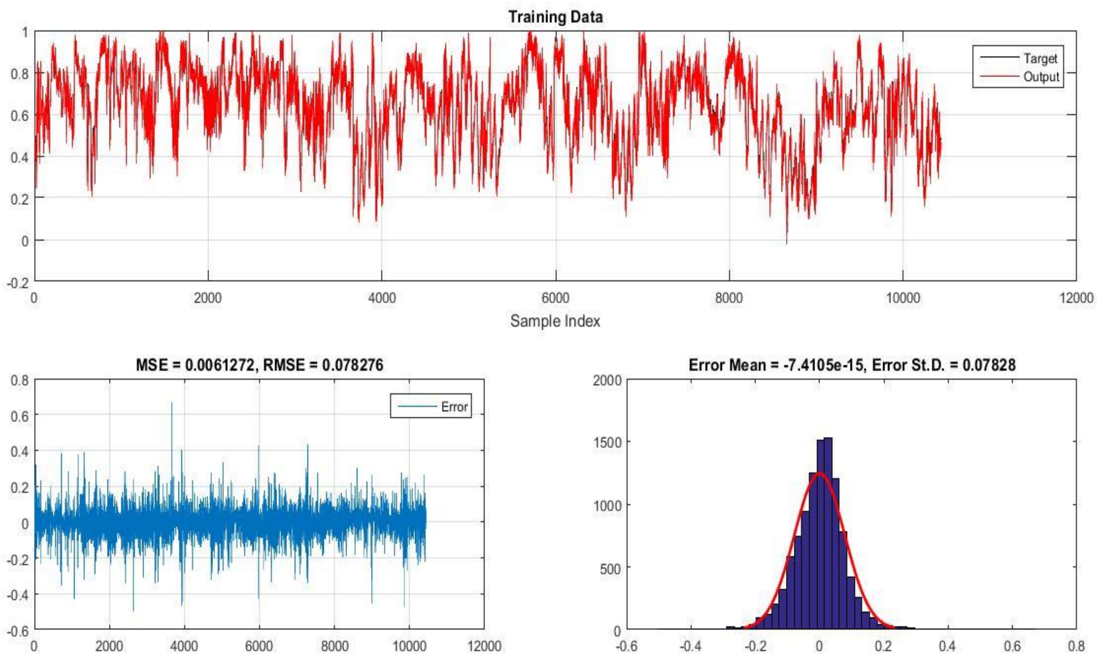


Figure 4.14. Training of Chicago Park Humidity Set using Handwritten .

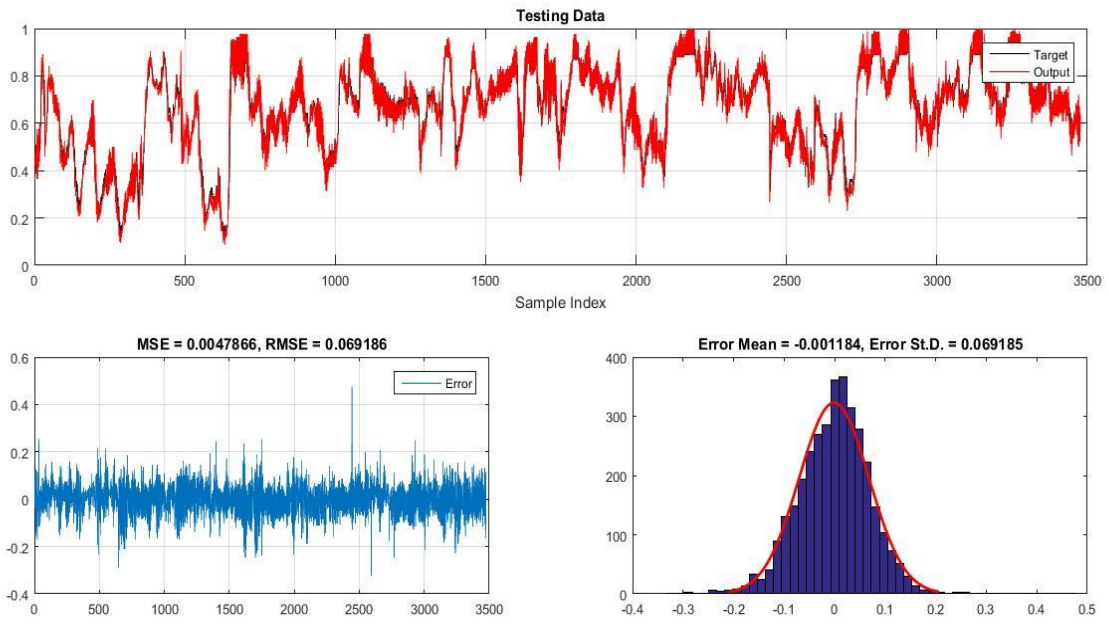


Figure 4.15. Testing of Chicago Park Humidity Set using Handwritten Algorithm.

While Figures 4.16 and 4.17 show the results of the same data set based on ANFIS-commands in Matlab Toolbox.

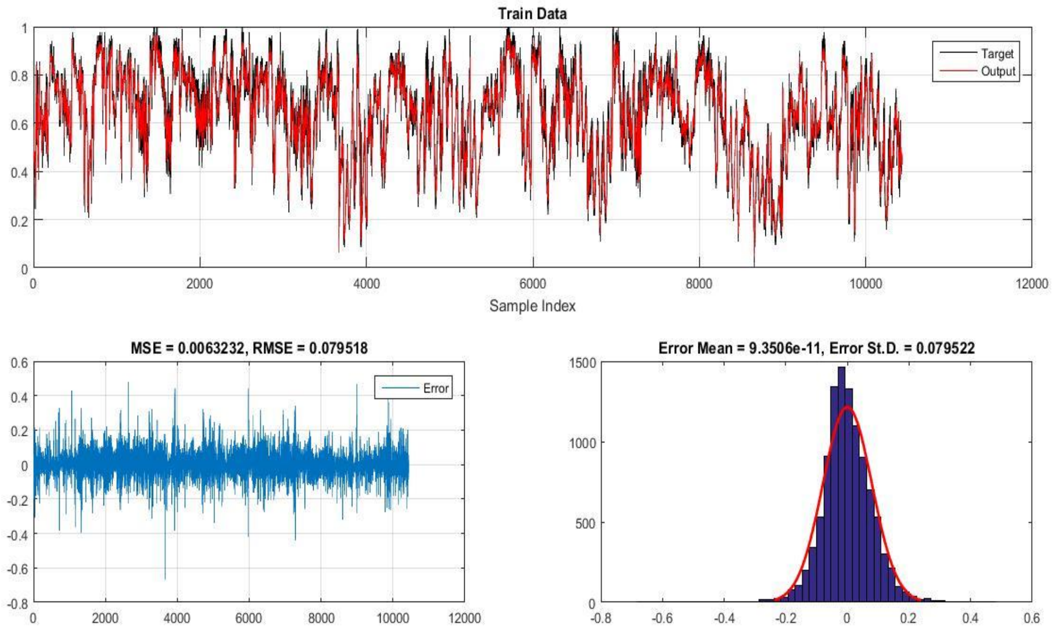


Figure 4.16. Training of Chicago Park Humidity Set using Matlab Toolbox.

Also, Table 4.4 summarizes all results **(**training 75% and testing 25%) in terms of mean error for humidity data set for both algorithms.

Table 4.4. Training and Testing results of Chicago Park Humidity data.

Technique type	Training	Testing
Matlab Toolbox	9.3506e-11	0.062509
Modified Algorithm	7.4105e-15	0.001184

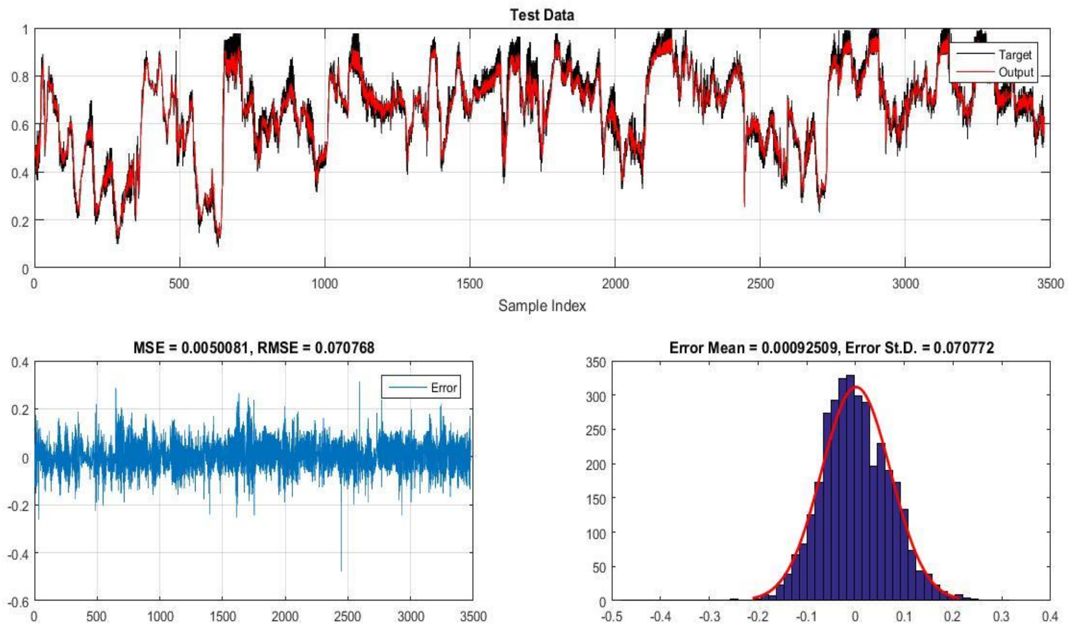


Figure 4.17. Testing Chicago Park Humidity Set using Matlab Toolbox.

Now, the comparison in both Table 4.3 and Table 4.4 summarize the results in all the above figures that are related to temperature and humidity data sets in terms of mean error. Integrating momentum factor and time series approach with handwritten algorithm provides very high accuracy in both training and testing phases compared with ANFIS-commands in Matlab toolbox.

Also, in terms of speed (time of execution), the modified algorithm minimizes the time execution that required for training algorithm. Adding momentum factor to the ANFIS algorithm helps to reduce the training time about 3.2-time from 3.675 to 1.1921 (in seconds) for temperature data set and from 3.6427 to 1.1895 (in seconds) for humidity data set. Table 4.5 demonstrates the time execution based on two algorithms.

Table 4.5. Training Speed (Execution Time) in Sec.

Execution Time in Sec (Training)		
Data Type	ANFIS Toolbox	Modified Algorithm
Temperature	3.675	1.1921
Humidity	3.6427	1.1895

While Table 4.6 shows the time of execution that is needed for testing both algorithms. According to the table, the time execution for both algorithms is almost the same. The time of execution equality in both algorithms is happened because of in the test phase, there is no need for any update parameters or minimize the overall error inside the algorithms. So, the test algorithms take a little time.

Table 4.6. Testing Speed (Execution Time) in Sec.

Execution Time in Sec (Testing)		
Data Type	ANFIS Toolbox	Modified Algorithm
Temperature	0.0124	0.011
Humidity	0.0122	0.012

Finally, Figure 4.18 and Figure 4.19 show the effective momentum factor on the training our modified algorithm by using temperature data set of Chicago park. According to the figures, the mean square error with momentum factor is (0.9058) after 40 iterations, while as equal (23.0417) after the same number of iterations without adding momentum factor to the ANFIS algorithm.

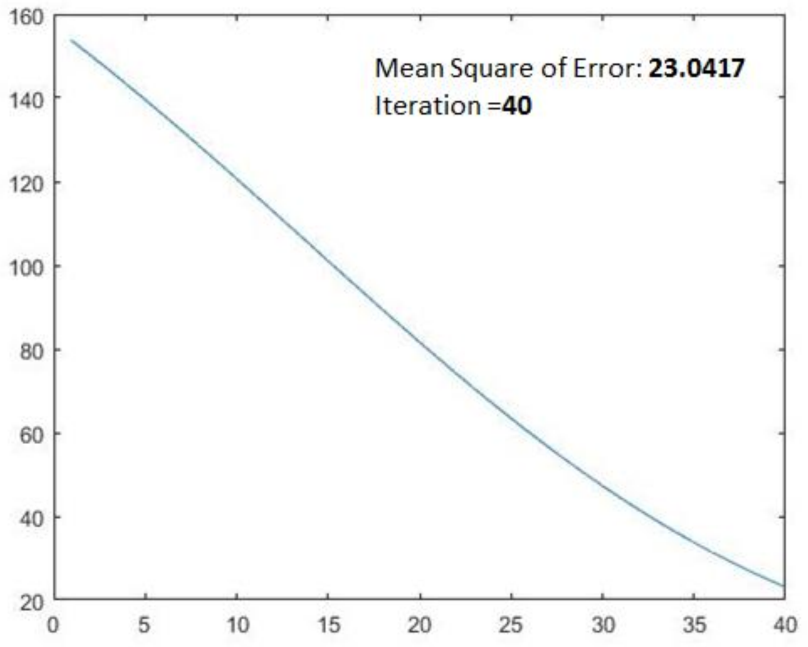


Figure 4.18. Training Modified Algorithm Without Momentum.

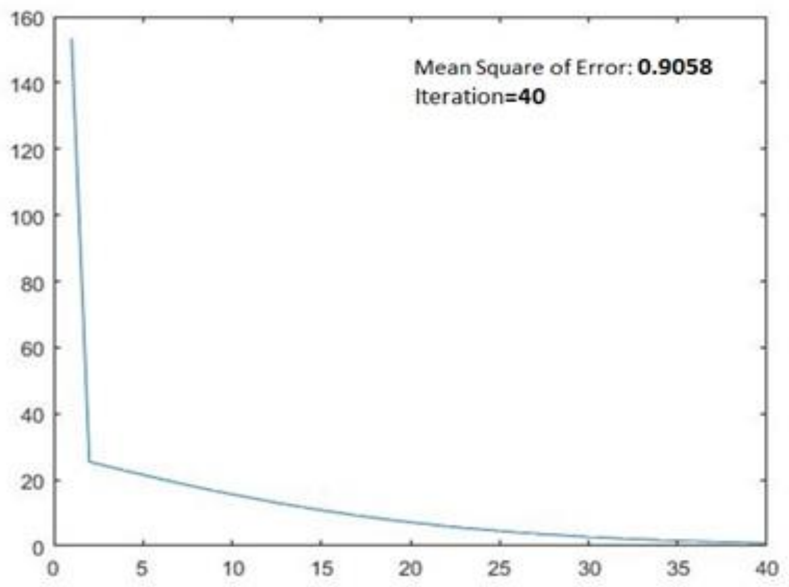


Figure 4.19. Training Modified Algorithm with Momentum.

4.4. Extended Architecture of ANFIS Algorithm

As we said in section 3.10.2.2, the ANFIS algorithm is trained by hybrid training techniques algorithms which comprising gradient descent and least square algorithms. each technique adjusts its parameters in a separate phase. The number of parameters in each phase is not constant. It depends on the number of transfer functions that are used at each level.

In this contribution, we will focus on linear parameters in layer 4 that are adjusted by a linear least square algorithm (LSM). these Parameters are known as consequent parameters and can be defined based on Eq. 4.2.

$$Q_4 = p_i x + q_i y + r_i \quad , \; i = 1,2,3, \qquad (4.2)$$

Where "**Q₄**" is the output of layer 4, and "**p**", "**q**" and "**r**" are adaptive consequent parameters.

The number of consequent parameters is defined based on the number of rules that identify by the user according to the complexity of the application. Figure 4.20 shows an example of ANFIS which has 9-rules in layer 3. Each rule is associated with one linear equation with a 3-adjustable coefficient. So, we have a 27-linear coefficient that should adapt during training iterations.

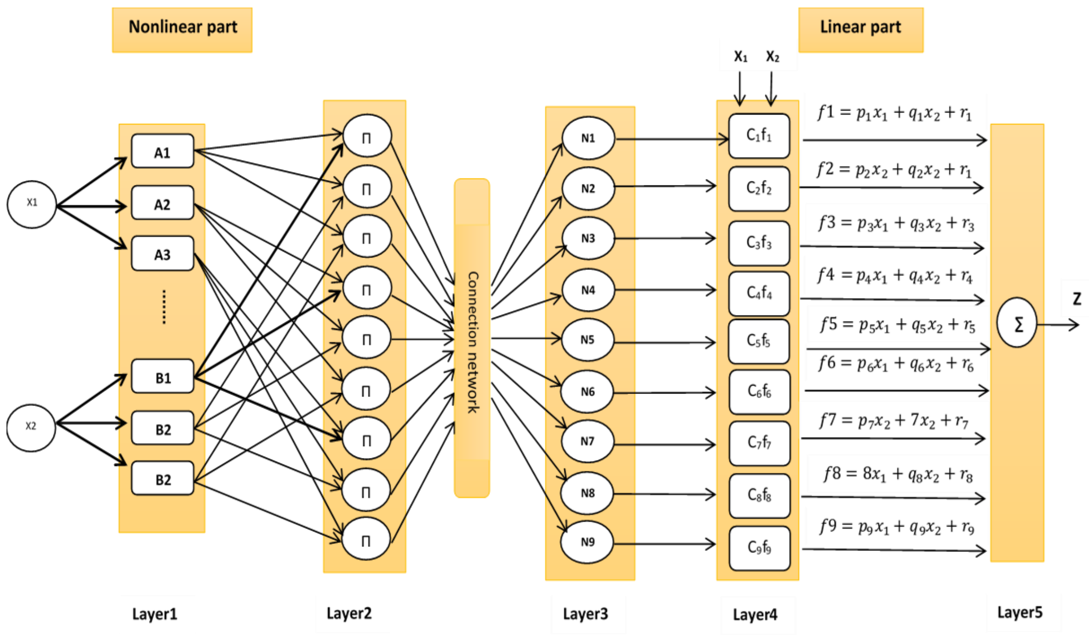

Nonlinear part
Linear part
X1
X2
A1
A2
A3
B1
B2
B2
Π
Connection network
N1
N2
N3
N4
N5
N6
N7
N8
N9
X₁
X₂
C₁f₁
C₂f₂
C₃f₃
C₄f₄
C₅f₅
C₆f₆
C₇f₇
C₈f₈
C₉f₉
$f1 = p_1x_1 + q_1x_2 + r_1$
$f2 = p_2x_2 + q_2x_2 + r_1$
$f3 = p_3x_1 + q_3x_2 + r_3$
$f4 = p_4x_1 + q_4x_2 + r_4$
$f5 = p_5x_1 + q_5x_2 + r_5$
$f6 = p_6x_1 + q_6x_2 + r_6$
$f7 = p_7x_2 + 7x_2 + r_7$
$f8 = 8x_1 + q_8x_2 + r_8$
$f9 = p_9x_1 + q_9x_2 + r_9$
Σ
Z
Layer1
Layer2
Layer3
Layer4
Layer5

Figure 4.20. ANFIS Algorithm with 9-Rules.

Another example of ANFIS algorithm that has 24-Rule associated with 81-linear coefficient can be found in Figure 4.21. We can easily notice that increasing the number of fuzzy rules led to increasing the number of linear parameters in layer 4 of ANFIS algorithm.

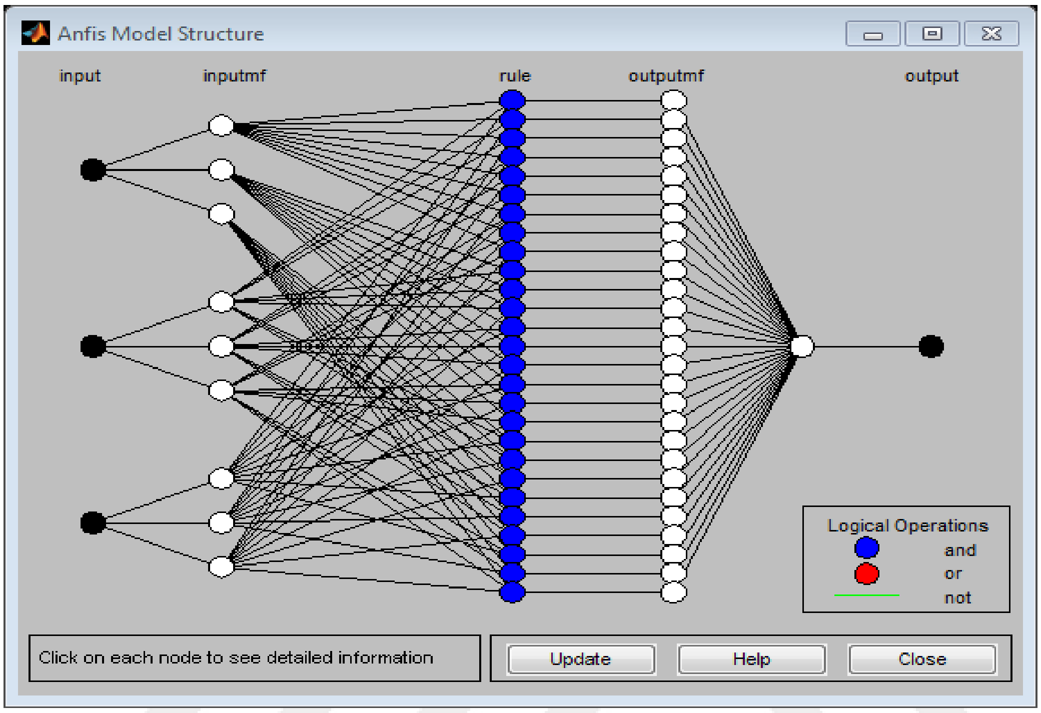


Figure 4.21. ANFIS Algorithm with 24-Fuzzy Rule.

Increasing the number of nodes in layer 2 (Figure 4.20) will help to achieve more accuracy but at the same time, it is a method to waste the system resources and consume more execution time when implemented in real-time applications because of the large number of linear parameters in the next layers that should calculate and adjust during the training process especially when designing a hardware version of this algorithm.

For this reason, we proposed a method to minimize the number of adjustable parameters at layer 4. To do that, another normalization layer after layer 4 is added to compress and minimize the number of linear equations without effect on the performance of the system. Since Figure 4.20 has 9-rules and 9-linear equations with 27-parameters, the normalization weight which is generated by layer 3 will be separated into several parts based on the node numbers in the following layer (layer 4). As an example, we will put 4-nodes in the new normalization layer. Therefore, the original weight in layer 3 will be divided into 4-part and each part will perform the aggregation process of all weights**.** Figure 4.22 illustrates this idea.

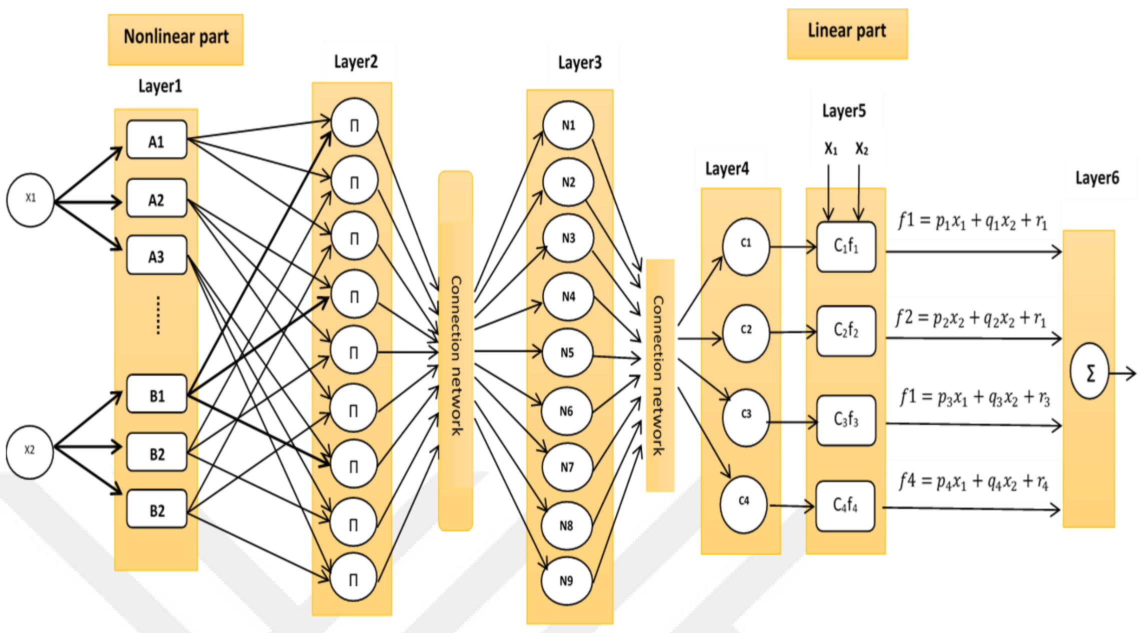


Figure 4.22. A Modified ANFIS Algorithm with a New Normalization Layer.

According to Figure 4.22, we have only 4-linear equations with 12-coefficient instead of 9-linear equations with 27-coefficient in Figure 4.20. We can use the equation Eq. 4.3 to calculate the output of the next layer.

$$w_c = \sum_i \overline{w}_{ij} \tag{4.3}$$

Where:

$w_{c:}$ is the new compressed weight,

i: is the number of nodes in the new proposed layer and,

j: is the number of nodes in the previous layer (layer 4).

As a result, we minimize the number of operations at out the last two layers (layer 5 and 6 in Figure 4.22), reduce the time of execution, and hardware resources when implementing it in real-time environments.

4.4.1. Test the Performance of Extended ANFIS Architecture

The evaluation of performance and speed of modified ANFIS in training and testing phases are done in this section. To training and testing the algorithm, humidity and temperature data sets for Izmir city/Turkey for 10-years ranging from 01/10/2006 to 31/12/2016 have been downloaded based on WU weather underground website. Each set of temperature and humidity consist of three groups. Each group has 365 elements as shown below:

- Sets of maximum values of temperature and humidity for each day.
- Sets of minimum values of temperature and humidity for each day.
- Sets of average values of minimum and maximum of both temperature and humidity for each day.

Now, we will use the ANFIS structure in Figure 4.23 to test the performance of the extended algorithm in terms of execution time and mean square error.

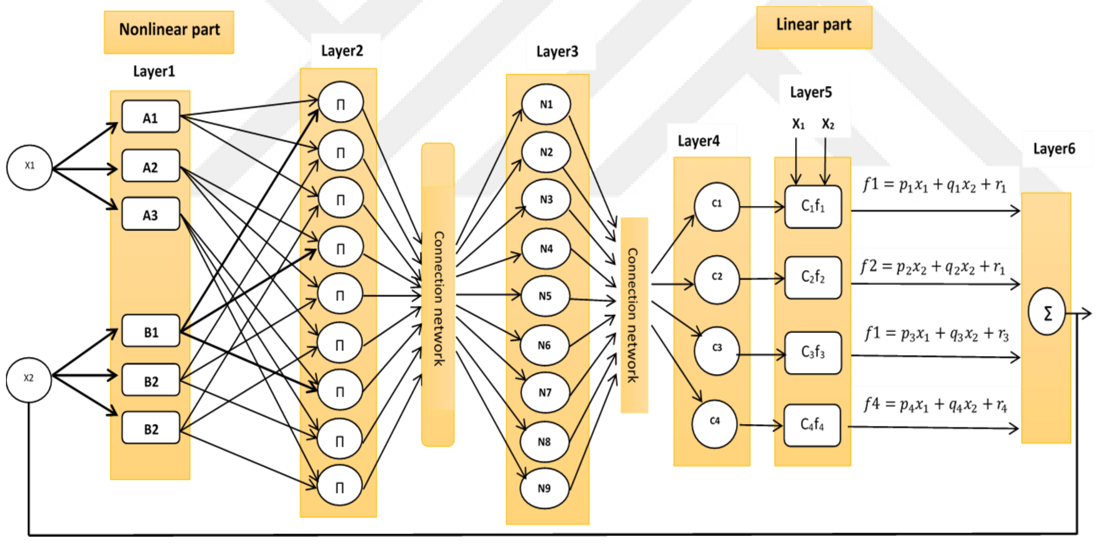


Figure 4.23. Extended ANFIS Architecture based on Recurrent .

First, sets of average values of both temperature and humidity are used to train the modified algorithm and then, the testing of the algorithm is done based on both maximum and minimum sets of temperature and humidity information. All results are compared with ANFIS-commands in Matlab toolbox. In the section below, we will list the training and testing results based on maximum values of temperature and humidity sets, while minimum values of temperature and humidity data sets will be put in the appendix A.

4.4.1.1. Training the Modified Algorithm and ANFIS Toolbox

The average of maximum temperature and humidity data set from 01/10/2006 to 31/12/2016 is calculated in order to train the extended ANFIS algorithm. Figure 4.24 shows the Training Results for these data sets.

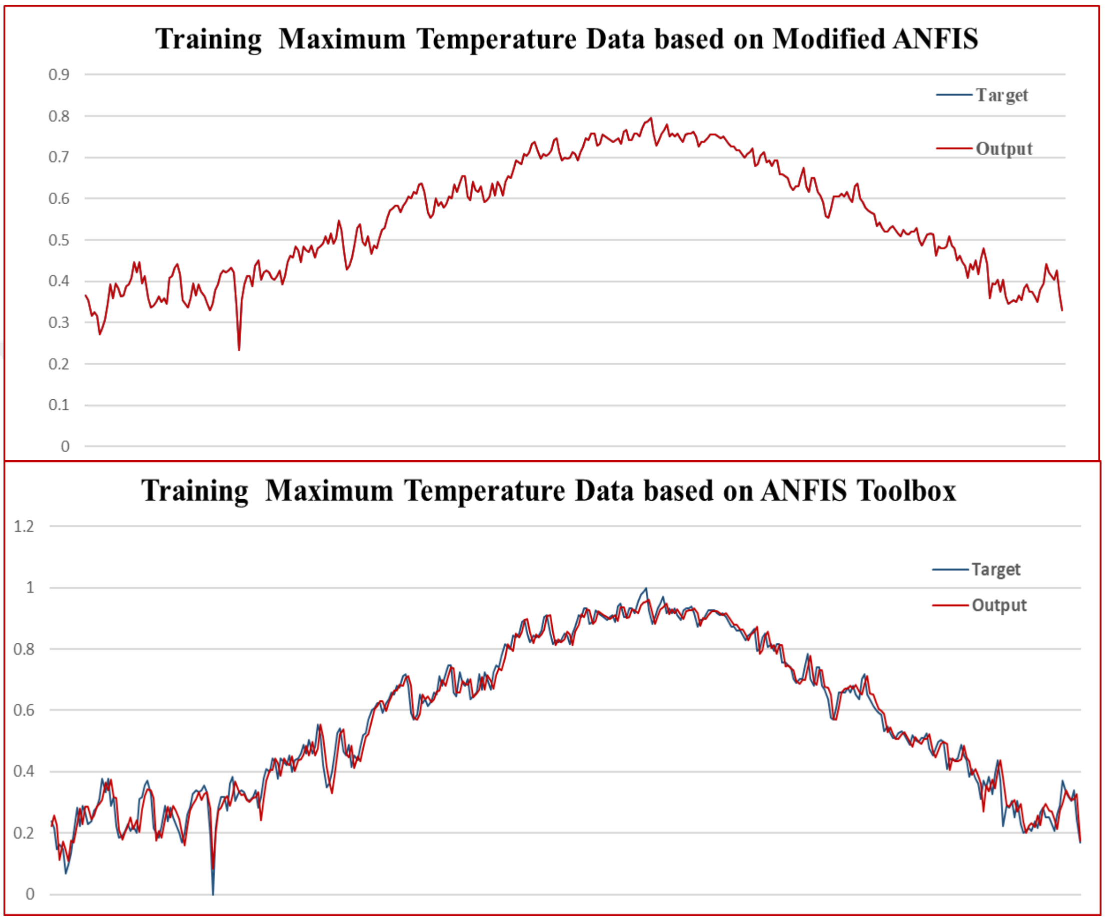


Figure 4.24. Training Temperature Set using Modified ANFIS and ANFIS Toolbox.

The results in Figure 4.24 represent the training of maximum temperature data set based on modified ANFIS (upper part) and ANFIS toolbox (lower part) and these results are achieved when using the following initial conditions for both algorithms:

- 3-Gaussian membership functions for each input.
- 9-Fuzzy rules that have a direct connection to equivalent normalization nodes.
- 4-linear equations with 12-adjustable linear coefficients.
- Adjustable momentum factor.
- 5000-training iterations.
- Small randum initial values for all parameters.

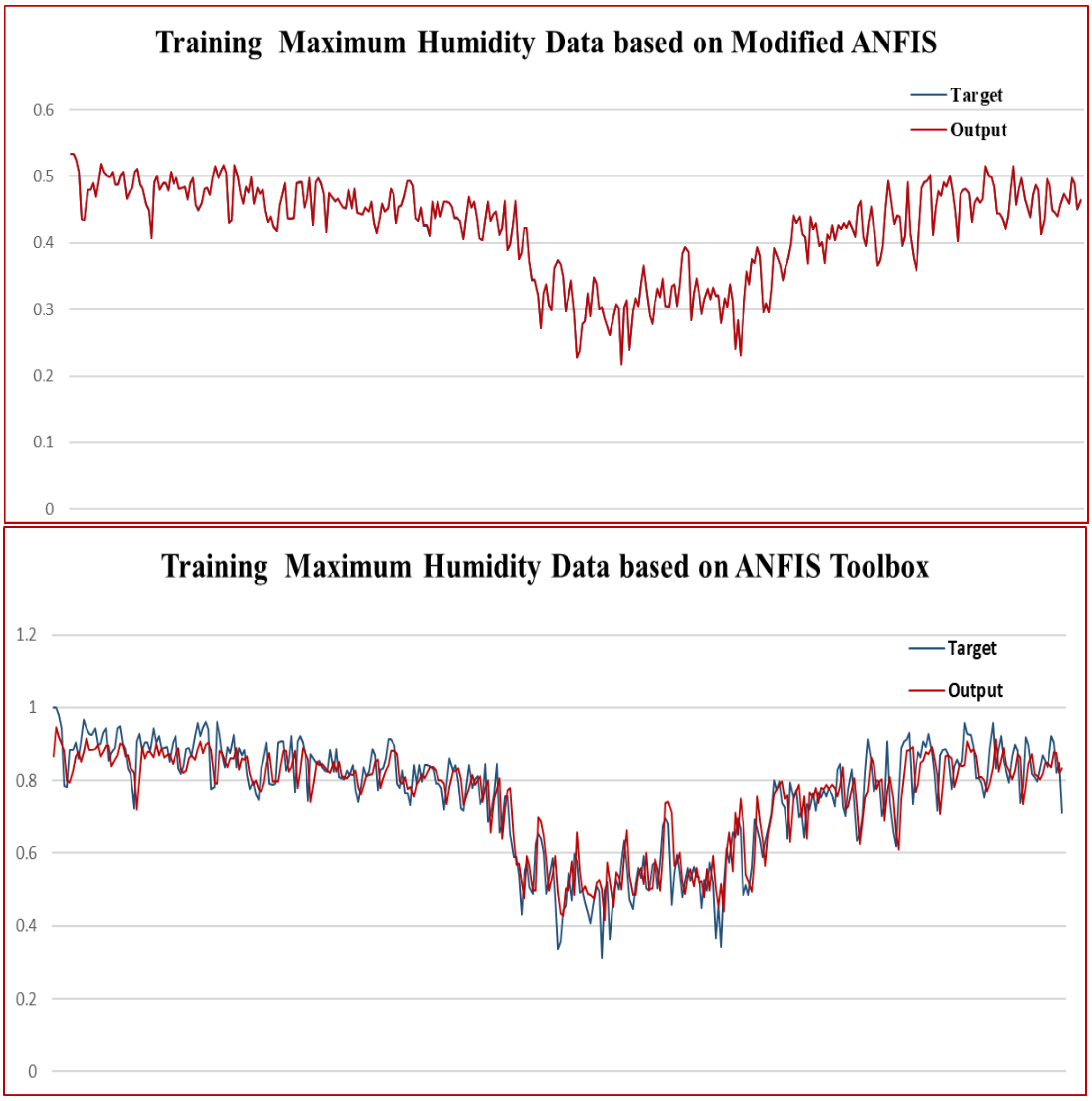
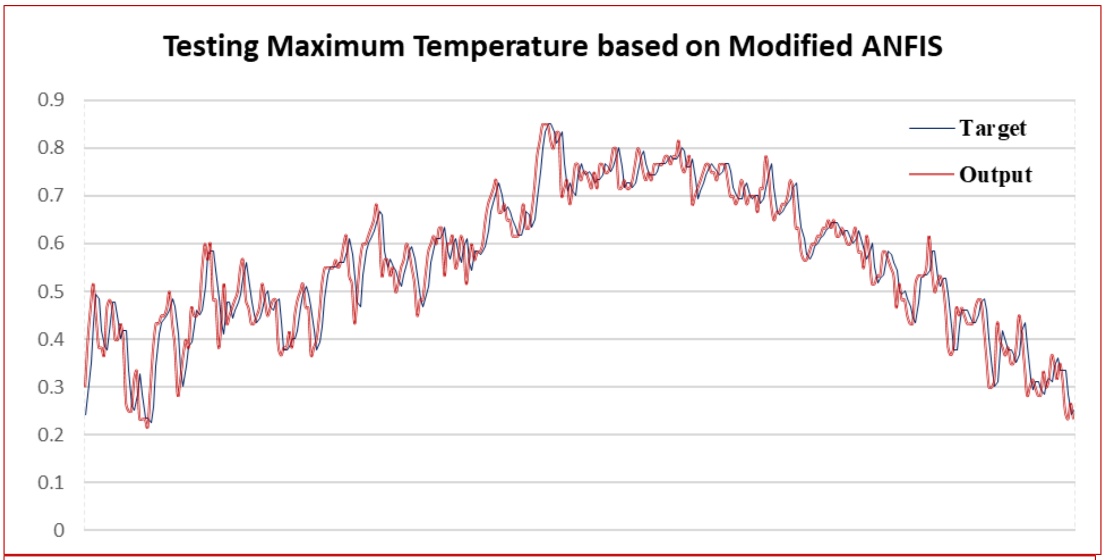


Figure 4.25. Training Humidity Data using Modified ANFIS and ANFIS Toolbox.

4.4.1.2. Testing Algorithms Using Maximum Temperature and Humidity Sets

The maximum temperature and humidity data sets are used to test the performance of the modified algorithm and compare the results with the ANFIS toolbox in Matlab. Figure 4.26 shows the testing results of the maximum temperature of the modified algorithm (upper part) and the ANFIS toolbox (lower part).



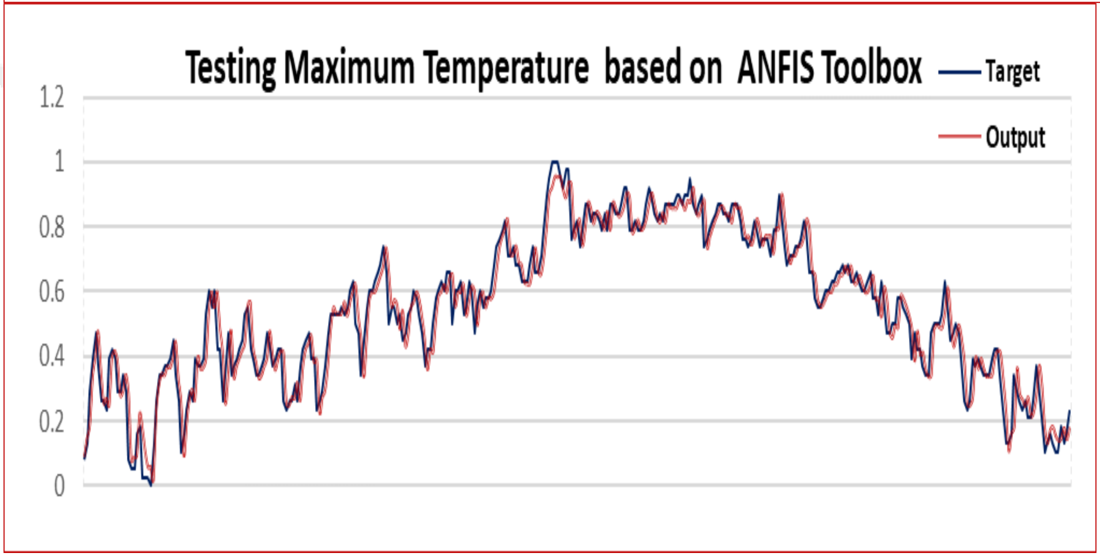


Figure 4.26. Testing Temperature Data using Modified ANFIS and ANFIS Toolbox.

While Figure 4.27 shows the testing results of the maximum humidity data set for both algorithm. All the above training and testing results are summarized and listed in Figure 4.28 in term of Mean Square Error (MSE). According to Figure 4.28, high accuracy is achieved by the modified algorithm compared with the ANFIS toolbox in both training and testing phases. Adding another layer in order to minimize the number of linear parameters does not affect the overall performance. Finally, the execution time of both training and testing phases based on the two methods are calculated in Figure 4.29 and Figure 4.30 respectively. The modified algorithm provides very high speed in the training phase compared with ANFIS toolbox and good speed in the testing phase.

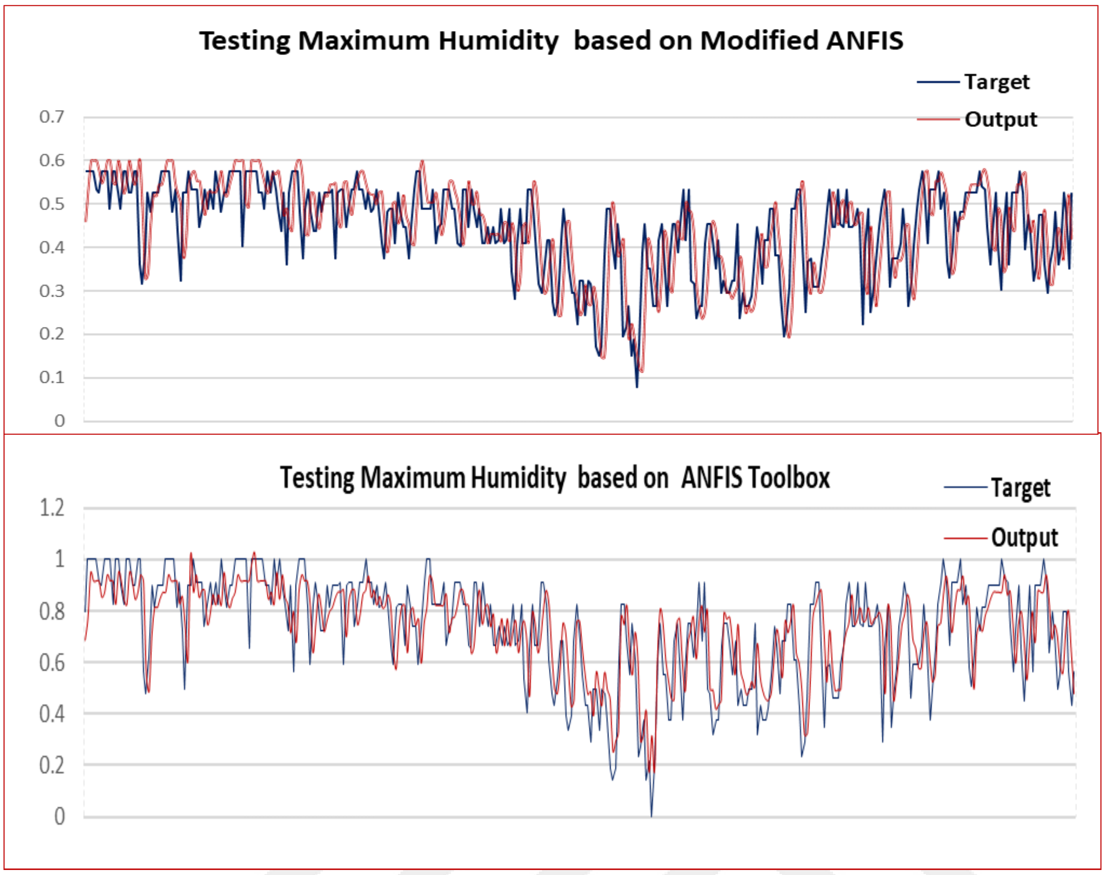
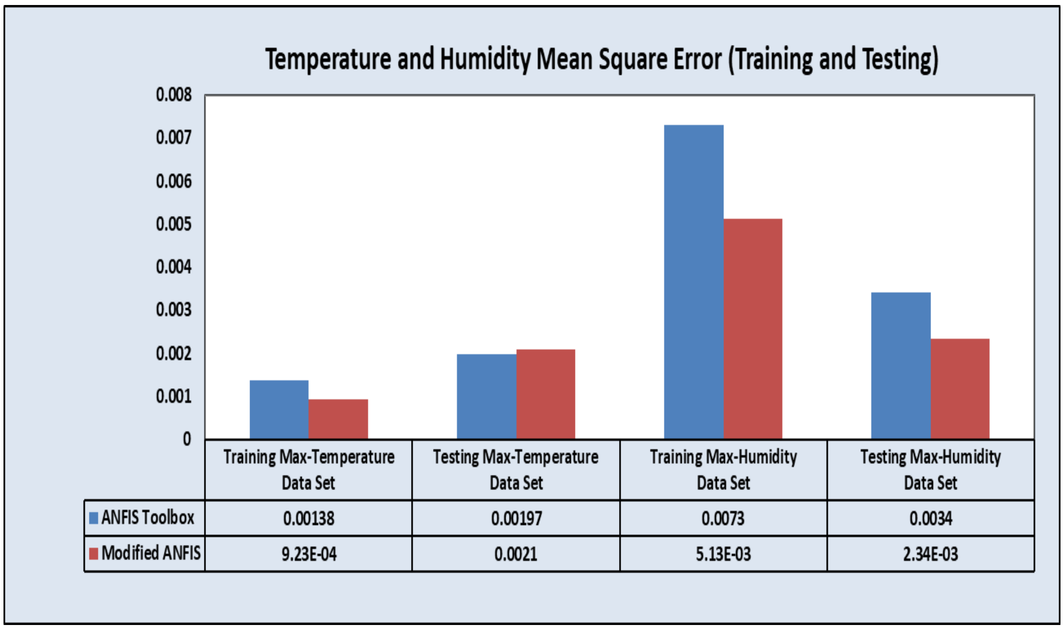
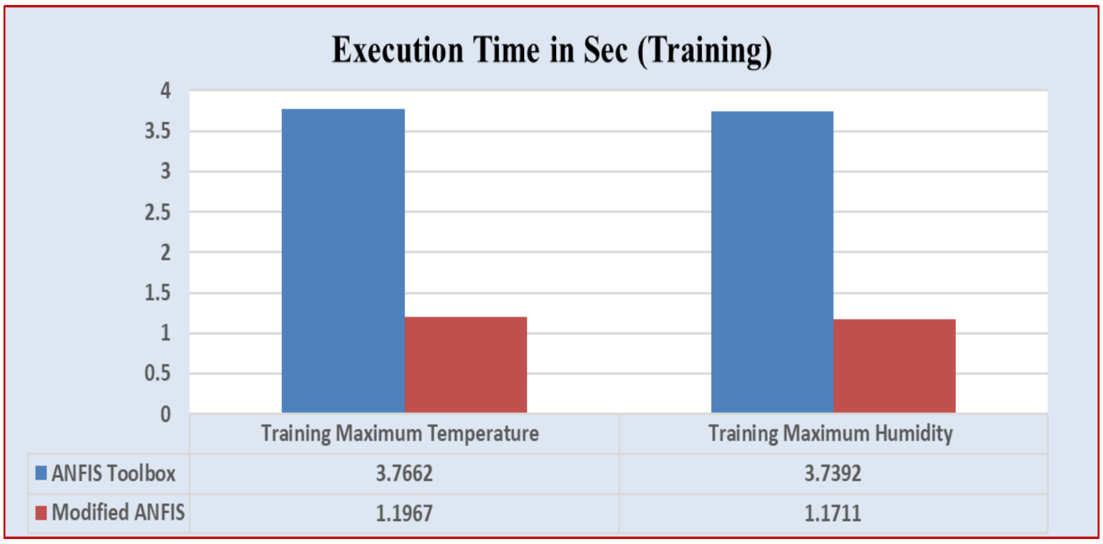


Figure 4.27. Testing Temperature Data using Modified ANFIS and ANFIS Toolbox



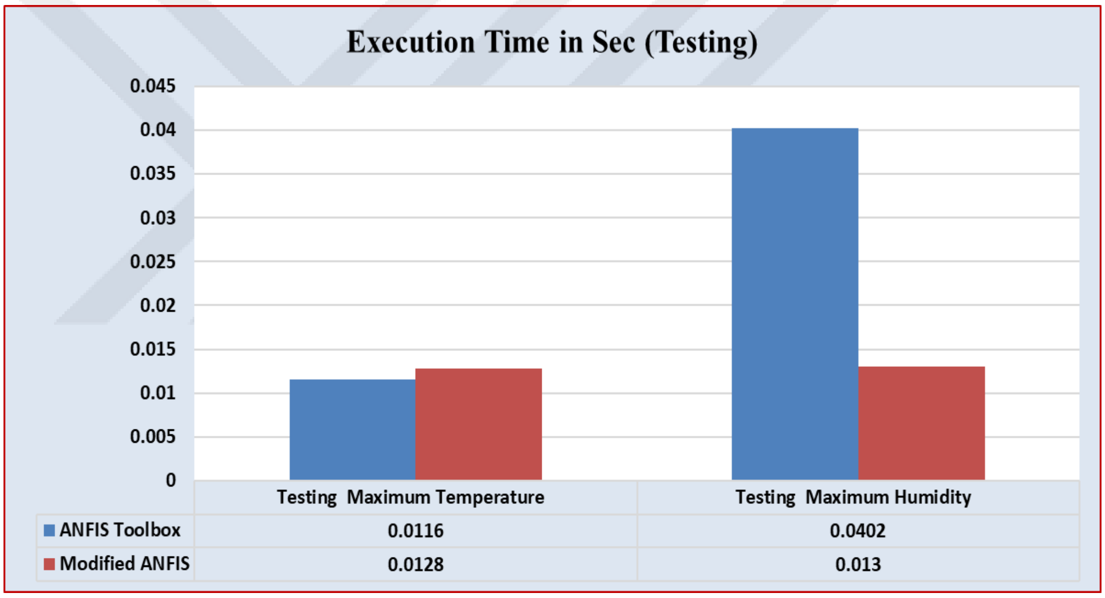
	Training Max-Temperature Data Set	Testing Max-Temperature Data Set	Training Max-Humidity Data Set	Testing Max-Humidity Data Set
🟦 ANFIS Toolbox	0.00138	0.00197	0.0073	0.0034
🟥 Modified ANFIS	9.23E-04	0.0021	5.13E-03	2.34E-03

Figure 4.29. MSE of Training /Testing of Maximum Temperature and Humidity.



	Training Maximum Temperature	Training Maximum Humidity
🟦 ANFIS Toolbox	3.7662	3.7392
🟥 Modified ANFIS	1.1967	1.1711

Figure 4.30. Execution Time of Training Phase.



	Testing Maximum Temperature	Testing Maximum Humidity
🟦 ANFIS Toolbox	0.0116	0.0402
🟥 Modified ANFIS	0.0128	0.013

Figure 4.31. Execution Time of Testing Phase.

CHAPTER 5
HARDWARE IMPLEMENTATION

5.1. Introduction

In this chapter, we will introduce the hardware implementation of single and multiple cores of modified ANFIS algorithm in the FPGA device. Also, we will introduce a short introduction about FPGA evaluation kit and the required software tools that used to implement the modified algorithm.

5.2. Hardware tools

In general, there are three types of FPGA tools that are used to implement any algorithm and design in the FPGA environment. In this section, we are going to give a short definition of each of them.

5.2.1. Xilinx Vivado Design Suite

For synthesis (compile) and perform analysis of any FPGA design (HDL design), Xilinx has presented a Vivado Design Suite. This software tool is a kind of an Integrated Design Environment (IDE). Vivado Design Suite helps developers and designers to perform several tasks such as system time analysis, Register Transfer Level (RTL) diagrams evaluation, design simulation and program the target FPGA device. Also, the software provides a tool for compiling and validating C-based algorithmic hardware cores (IP) based on Electronic System Level (ESL) that provides a design abstraction and early design implementation to enable wrapping and integrating both RTL IP and algorithmic-based IP for reuse (Xilinx Inc. SDK, 2016). Figure 5.1 (Synthesis, 2016) shows the tasks that are done by Vivado Design Suite.

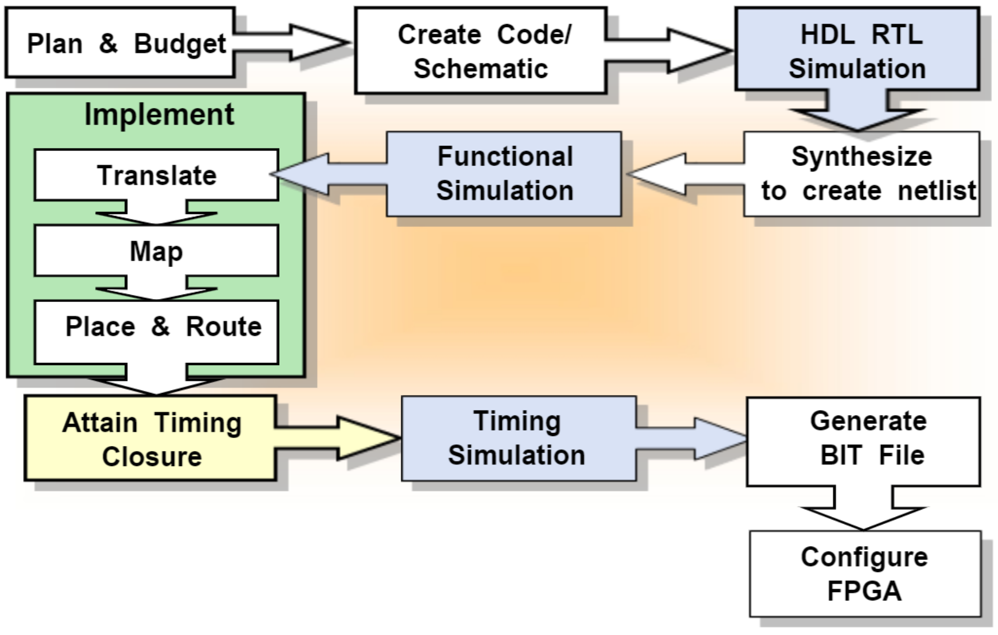


Figure 5.1. Xilinx Design Flow.

5.2.2. Xilinx Software Development Kit (SDK)

Building software platforms of Xilinx FPGA devices can be done by Xilinx Software Development Kit (SDK) tool. SDK tool deals with embedded hardware designs that are created based on Vivado design suite tool. Several advantages can be achieved by SDK tool such as (Xilinx Inc. SDK, 2016):

- Compilation environment and code editor based on C/C++ languages.
- Error detection.
- Project management.
- Automatic Make-file creation.
- Debugging and describing of embedded design.
- Source code control and management.

Figure 5.2 describes the necessary steps to build any software platform of Xilinx FPGA devices.

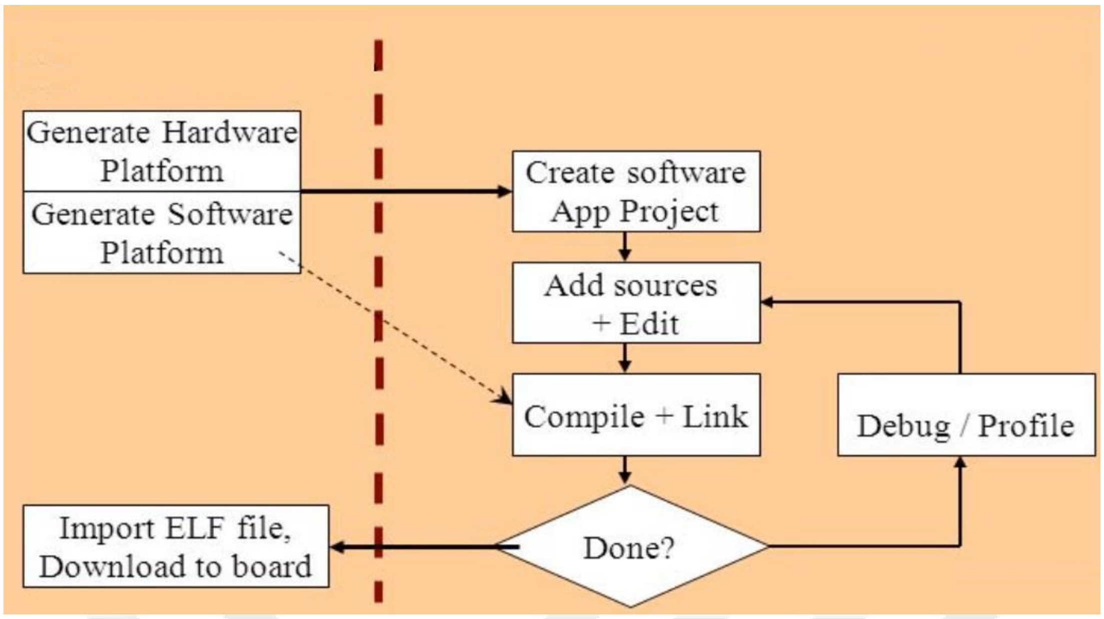

Generate Hardware Platform
Generate Software Platform
Create software App Project
Add sources + Edit
Compile + Link
Debug / Profile
Done?
Import ELF file, Download to board

Figure 5.2. SDK development Flow.

5.2.3. Vivado High-Level Synthesis (HLS)

High-Level Synthesis (Vivado HLS) from Xilinx is a software that is used to generate an IP core-based hardware design. This tool allows algorithms that are written by System C, C++, and C languages to be targeted into all Xilinx FPGA devices directly without any need to manually build RTL design(Synthesis, 2016).

5.3. Hardware Design

The modified ANFIS algorithm that was presented in Chapter 4, has been implemented on the seventh generation of Xilinx FPGA device. We utilized the Xilinx Zynq xc7z020 evaluation kit to be our hardware platform. To achieve our implementation, our project is divided into multi-stages:

- **First stage:** The first stage in our design implementation depends on Vivado High-Level Synthesis tool to build a specific soft IP core for the hardware version of our modified ANFIS-based time series algorithm based on a small C language that is suitable for programming limited hardware resource systems such as embedded systems and microcontrollers. In order to economize the hardware resources that are used by our IP core, we will use a 16-bit fixed point approach and some directive functions such as (data flow, loop merge, and pipeline) to optimizing the hardware ANFIS core. After that, we will export the hardware ANFIS core and associated files from Vivado HLS to the Vivado Design Suite in order to bind it to other hardware components in the FPGA board. Figure 5.3 shows a single ANFIS IP core (denoted by red rectangular) which is connected to other system components.

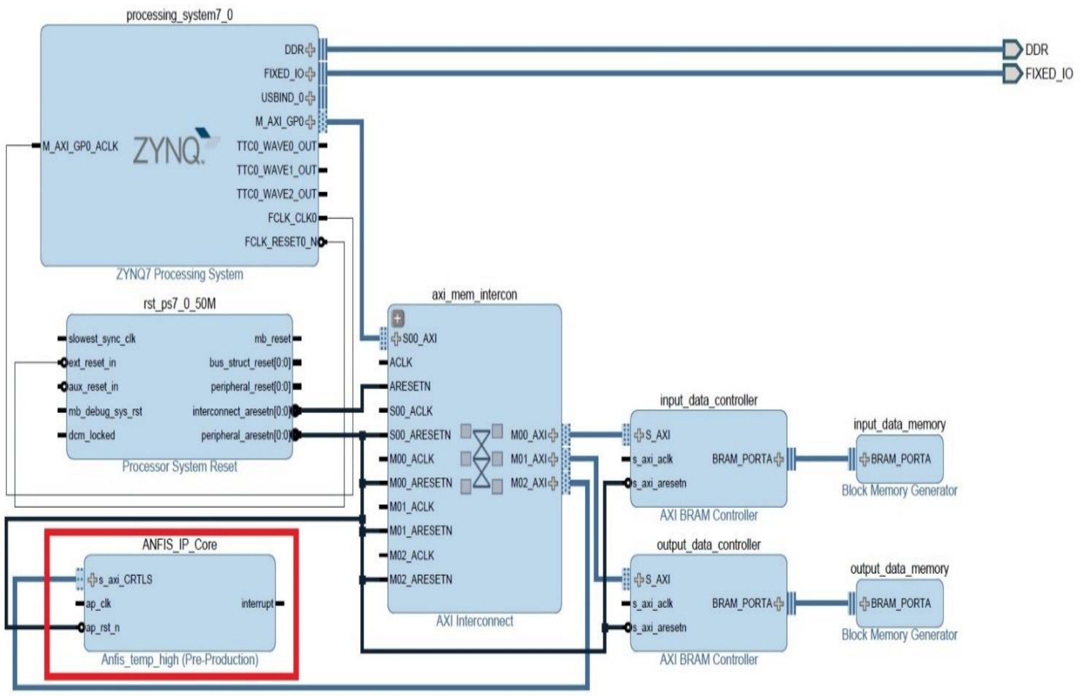
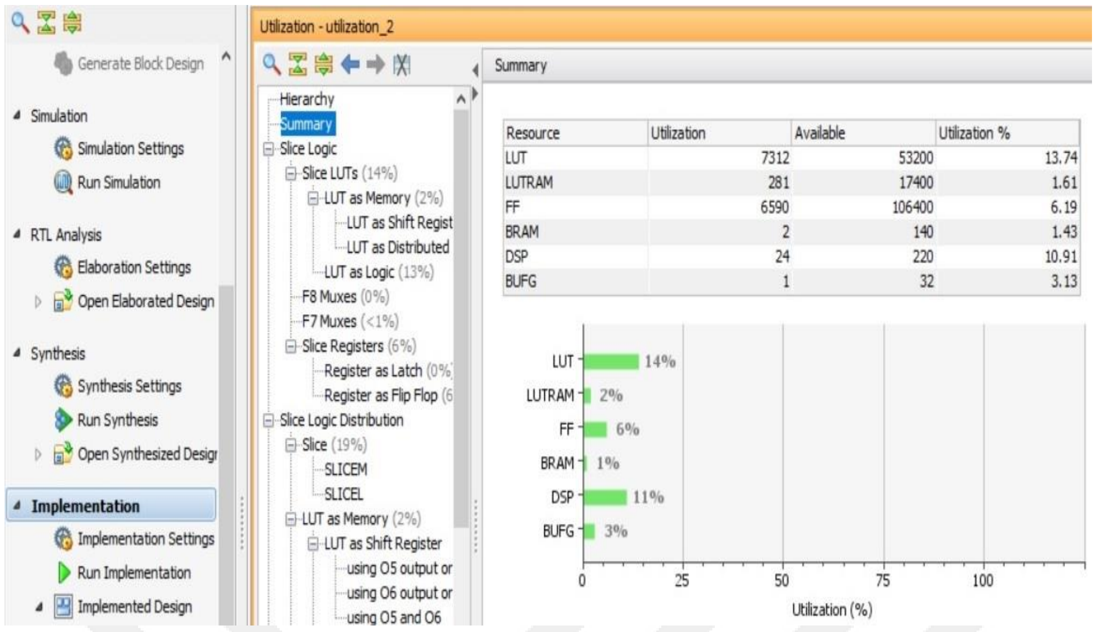


Figure 5.3. Hardware Design with a Single ANFIS IP.

According to Figure 5.3, several main components are required for our design to work correctly as shown below:

- Dual-core ARM Cortex-A9 as a main system processor.
- One soft IP core for modified ANFIS algorithm.
- One block of memory and its controller to store the data that come from sensors.
- One block of memory and its controller to store the data that will be sent to PC.
- Reset system block and system interconnections

Figure 5.4 describes utilization resources that required to build and connect a single ANFIS core in Zynq xc7z020 platform.



Resource	Utilization	Available	Utilization %
LUT	7312	53200	13.74
LUTRAM	281	17400	1.61
FF	6590	106400	6.19
BRAM	2	140	1.43
DSP	24	220	10.91
BUFG	1	32	3.13

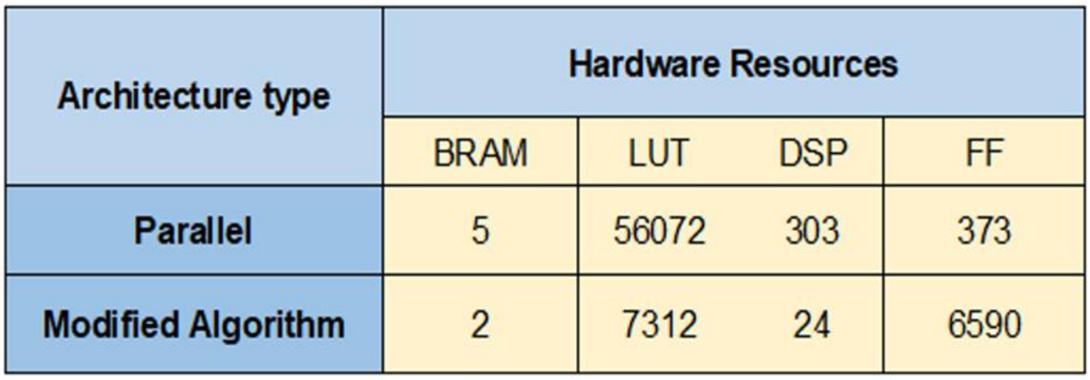
Figure 5.4. The Utilization Hardware Resources for Single ANFIS IP.

The Zynq xc7z020 evaluation kit comprises (Xilinx, 2018; Xilinx Inc. SDK, 2016) 53200-LUT, 106400-FF, 140-BRAM,32-BUFG and 220-DSP block. While the implementation needs:

- 14% (7312) of LUT (Lookup Table) and 2% of LUTRAM (RAM Lookup Table).
- 6% (6590) of FF (Flip-Flop).
- 1% (2) of BRAM (Block RAM).
- 11% (24) of DSP Block (Digital Signal Processor)
- 3% (1) of BUFG (Global Buffer)

Logic resources of modified ANFIS algorithm are compared with some existing parallel hardware structures of ANFIS algorithm (Brassai, Hajdu, Tamas, and Bakó, 2015; Mas and Brassai, 2015; Gómez-Castañeda, Tornez-Xavier, Flores-Nava, Arellano-Cárdenas, and Moreno-Cadenas, 2014) that implemented it in various FPGA devices. Table 5.1 lists hardware utilization resources for different approaches.

Table 5.1. Comparison of Hardware Resources for the Different Approaches.



Architecture type	Hardware Resources			
	BRAM	LUT	DSP	FF
Parallel	5	56072	303	373
Modified Algorithm	2	7312	24	6590

According to Table 5.1, our modified algorithm reduces the number of logic circuits that are used by modified ANFIS core like DSP block, size of RAM, and the number of LUT when compared with parallel ANFIS structures. Also, the proposed ANFIS core requires additional FF logic compared with the traditional parallel implementation. Additional numbers of FF logics that are used by modified algorithm provide high speed compared with other hardware implementation and more convenient for real-time applications.

- **Second stage:** Based on Xilinx Vivado Suite tool the overall hardware design has built. The ensemble of hardware ANFIS-IP cores in Figure 5.5 composes of 6-ANFIS cores. ANFIS cores have been trained to deal with contrastive dataset (minimum and maximum humidity and temperature values). Each core receives the real data from a specific connected sensor. In our implementation, we will connect 2-set of sensors (3-sensor for temperature and 3-sensor for humidity). Each sensor is connected to equivalent ANFIS core through external co-processor (we will explain this co-processor in the next chapter). So, the first 3-ANFIS cores are connected to different 3-humidity sensors while other 3-temperature sensors are connected to remaining 3-ANFIS cores. All received data from sensors will be stored in a temporary block RAM in order to perform pre-processing tasks and prepare these data to be suitable for each hardware core. Also, we will use one block RAM to store all results and prepare them to the next processing stage. Figure 5.5 illustrates this hardware design.

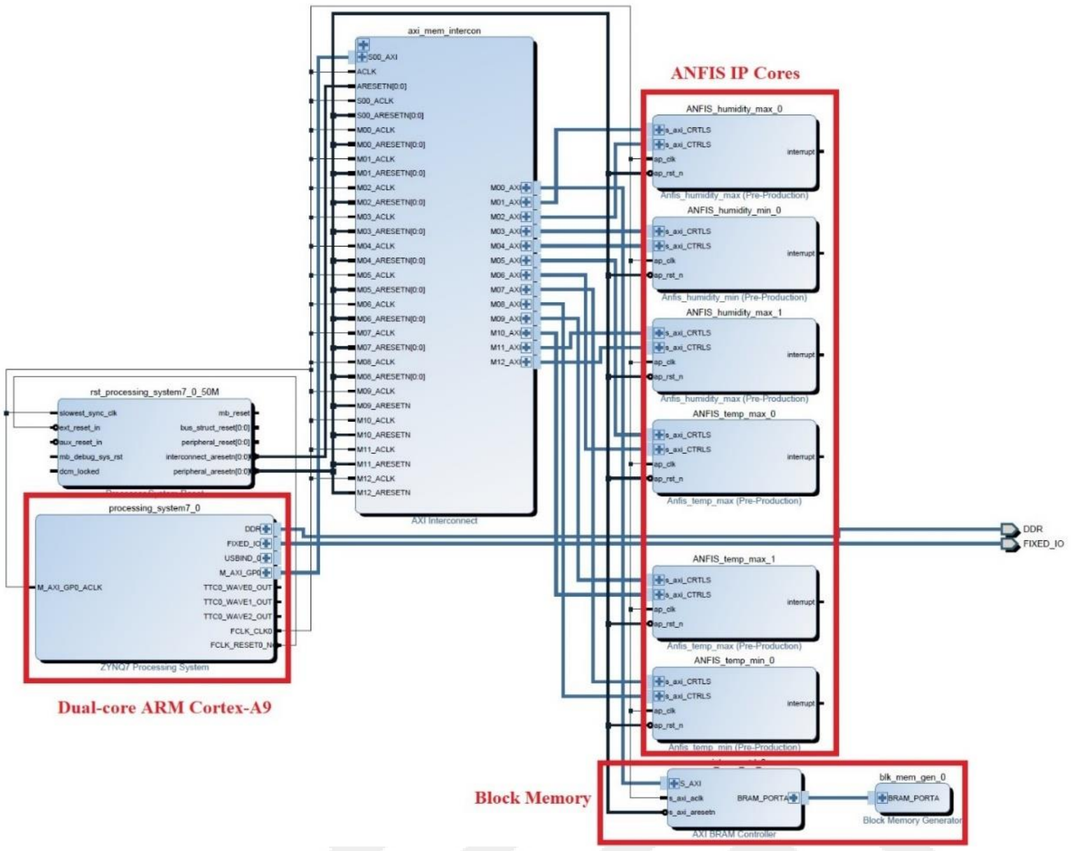
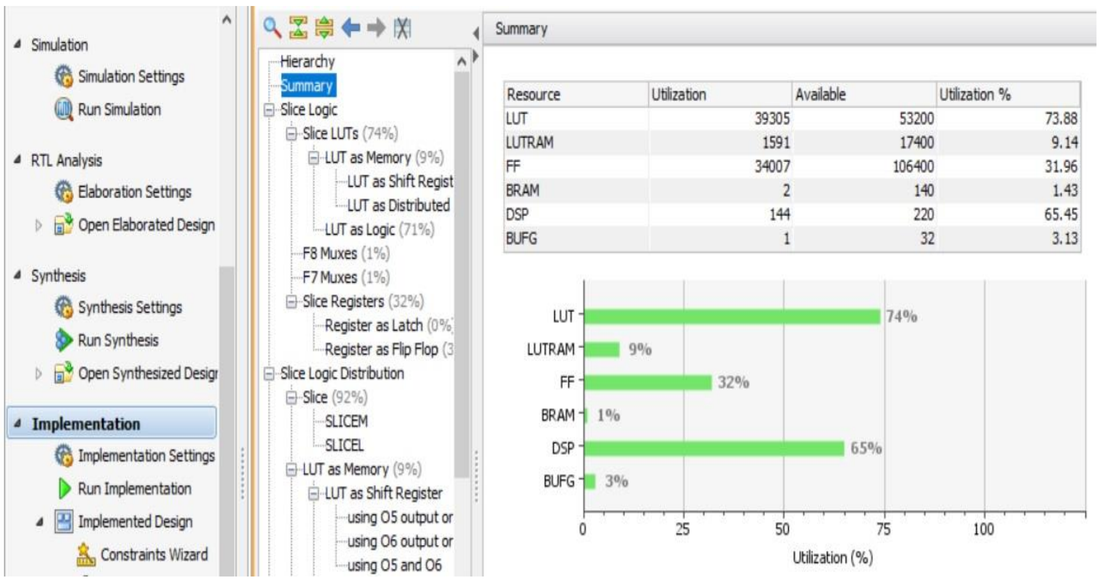


Figure 5.5. Hardware design with Multi-ANFIS IP core.

The hardware resources required for the design in Figure 5.5 are calculated in Figure 5.6.



Resource	Utilization	Available	Utilization %
LUT	39305	53200	73.88
LUTRAM	1591	17400	9.14
FF	34007	106400	31.96
BRAM	2	140	1.43
DSP	144	220	65.45
BUFG	1	32	3.13

Figure 5.6. Hardware Resources for Multi-ANFIS Cores.

- **Third stage:** This stage is done based on Xilinx Software Development Kit (SDK) tool. In this stage, all drivers, system parameters, and cores initialization of ANFIS algorithms are prepared in order to evaluate the performance of the system. All received data from Arduino co-processor are stored in system memory (Block Memory in Figure 5.5) and then, the ARM Cortex-A9 processor (main system processor) fetchs the data from block memory in order to handle the data pre-processing tasks. The data pre-processing can be divided into several sub-steps:

1. **Data classification:** All sensing real data (temperature and humidity) that are sent by co-processor are associated with the unique 8-bit ID. The main system processor performs the data classification task according to its ID using a basic classification algorithm (if-then condition approach). Then, the data that are classified are stored in different memory bank within the main system memory.

2. **Data normalization:** which scales all sensing data in the range [0,1]. In our thesis, the formula in Eq. 5.1 is used:

$$x_{new} = \frac{x_{input} - x_{min}}{x_{max} - x_{min}} \tag{5.1}$$

   Where x_{max} and x_{min} are the maximum and minimum values of the sensing data

3. **Data representation:** Cortex-A9 processor supports different data types such as floating point and fixed-point data representation. So, we need to convert all sensing data to unsigned data representation that acceptable by both IP cores and memory based on IEEE754 standard. In our case, the sensing data are converted from a 16-bit fixed point to unsigned 16-bit data representation.

4. **Data fuzzification:** To reduce the number of resources that are utilized by single ANFIS core, we split the layer one (fuzzification layer) from ANFIS algorithm and implement it within the Cortex-A9 processor instead of ANFIS core. Splitting the ANFIS tasks between ANFIS core and Cortex-A9 plays a key role to reduce the number of resources that are used by the Zynq programming logic (PL) and use the power of Cortex-A9 processor in Zynq processing system (PS). This step

decreases the number of DSP block by 55% when compared with (Mas & Brassai, 2015) that only works which used a similar approach.

5. **Data routing:** during this step, the Cortex-A9 processor forwards the classified data to equivalent ANFIS cores and then transfers results to the PC via Ethernet controller. Figure 5.7 illustrates the above processing steps.

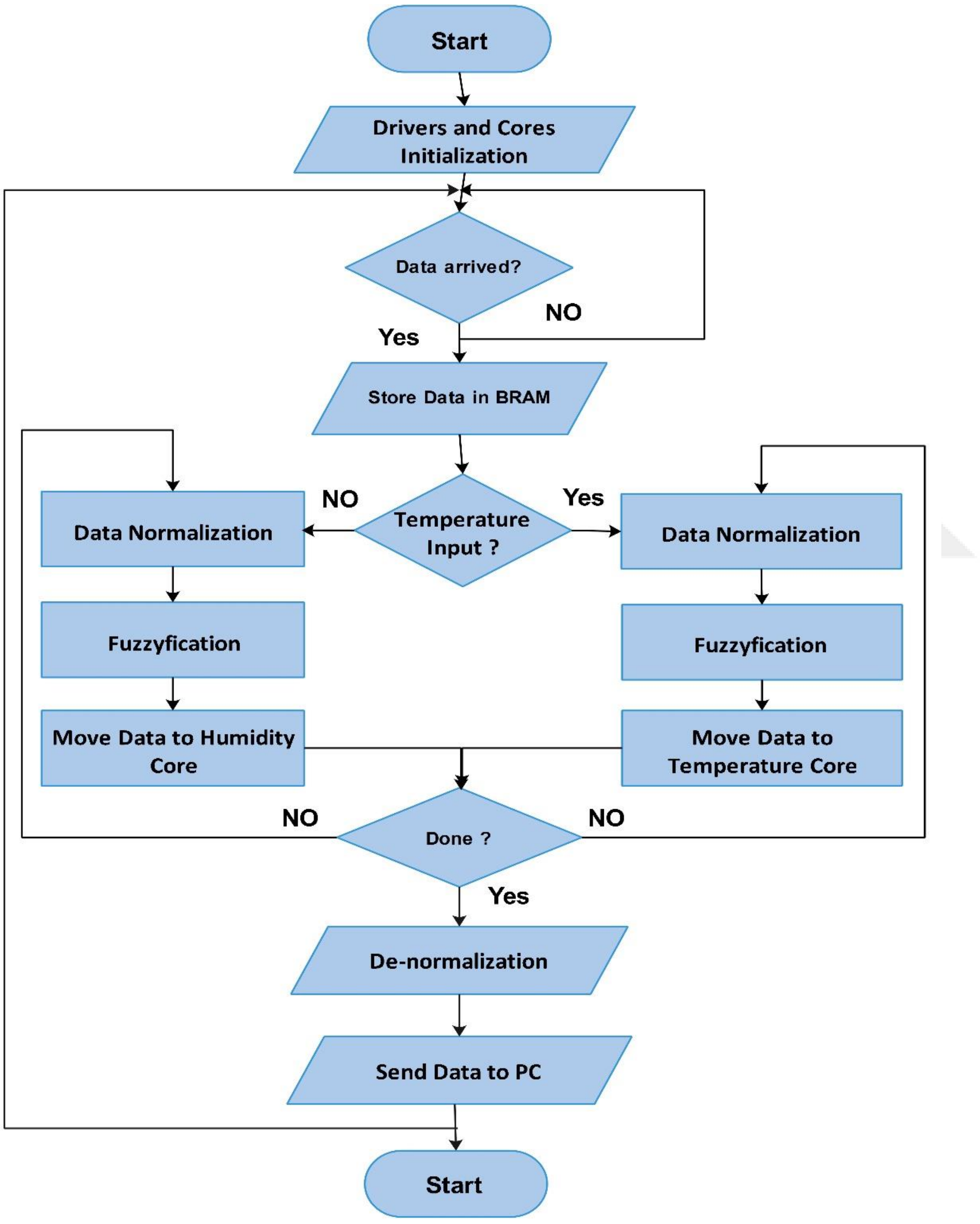


Figure 5.7. Processing Steps that Implemented by Cortex-a9 Processor.

5.4. Hardware Test and Results

Matlab Implementation results that are obtained in chapter 4 are used to test evaluate and compare the performance of the hardware design with the ANFIS implementations in Matlab environment (software version) in term of mean square error (MSE) and mean of error. Figure 5.8 and Figure 5.9 show comparison between hardware and software result of both maximum temperature and humidity data respectively.

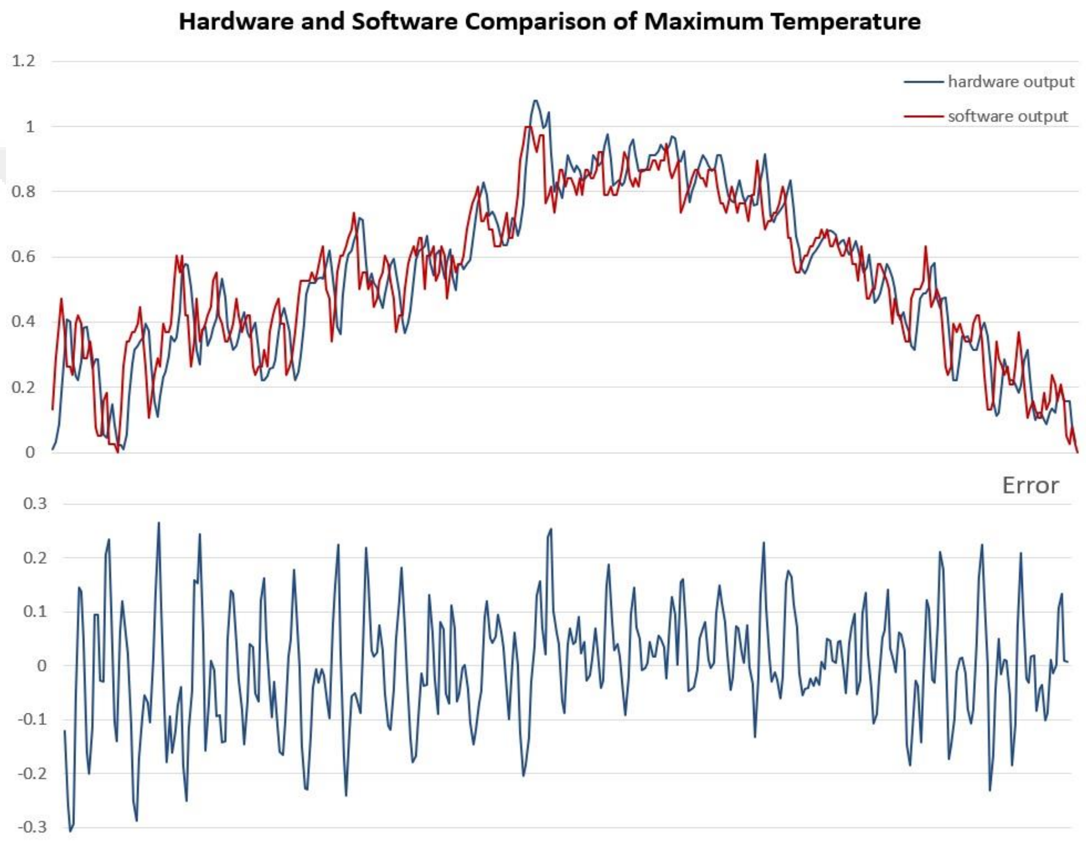


Figure 5.8. Hardware and Software Comparison of Maximum Temperature.

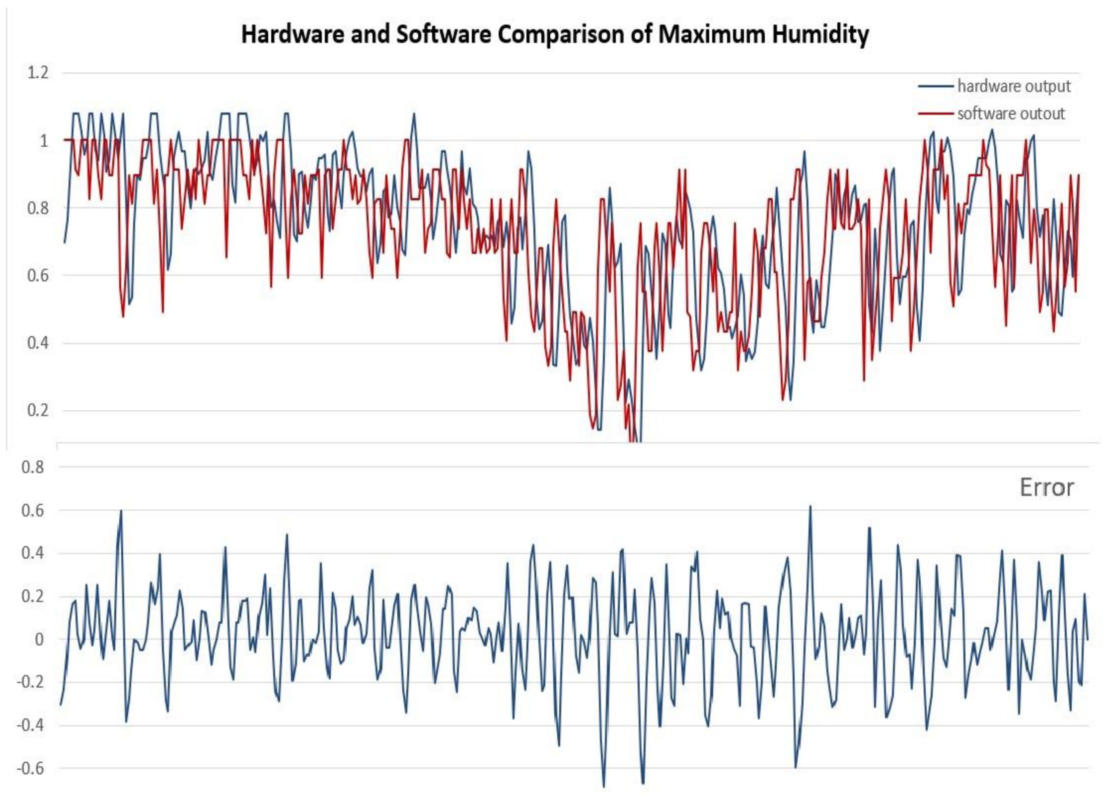


Figure 5.9. Hardware and Software Comparison of Maximum Humidity.

Figure 5.10 and Figure 5.11 show comparison between hardware and software result of both minimum temperature and humidity data respectively.

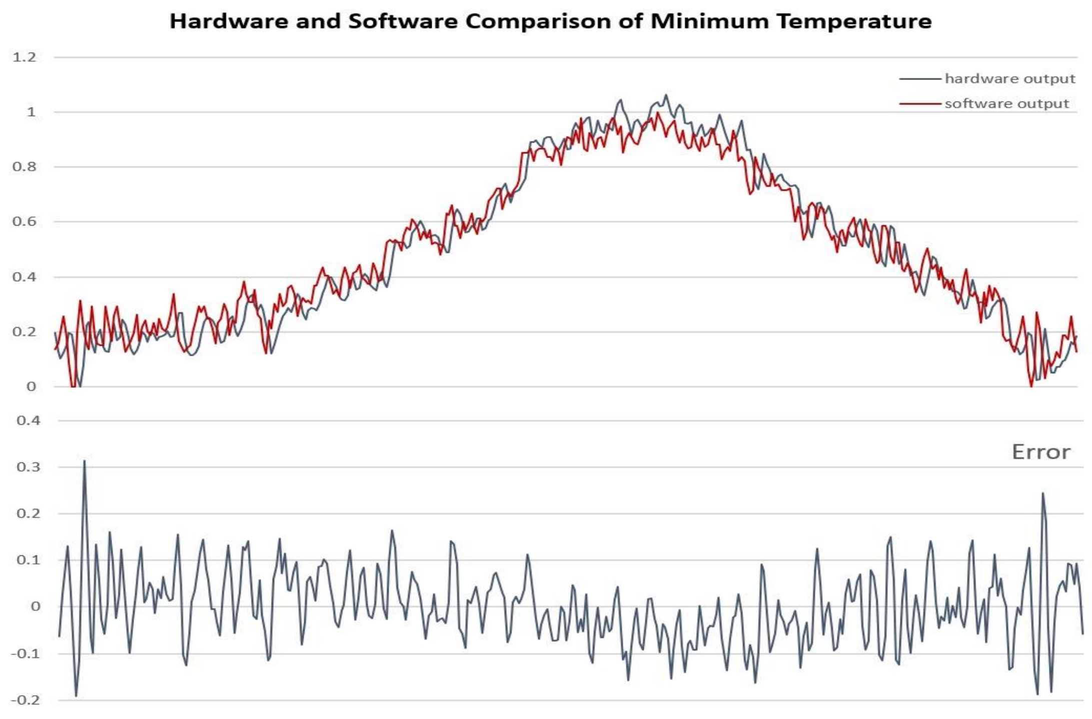


Figure 5.10. Hardware and Software Comparison of Minimum Temperature.

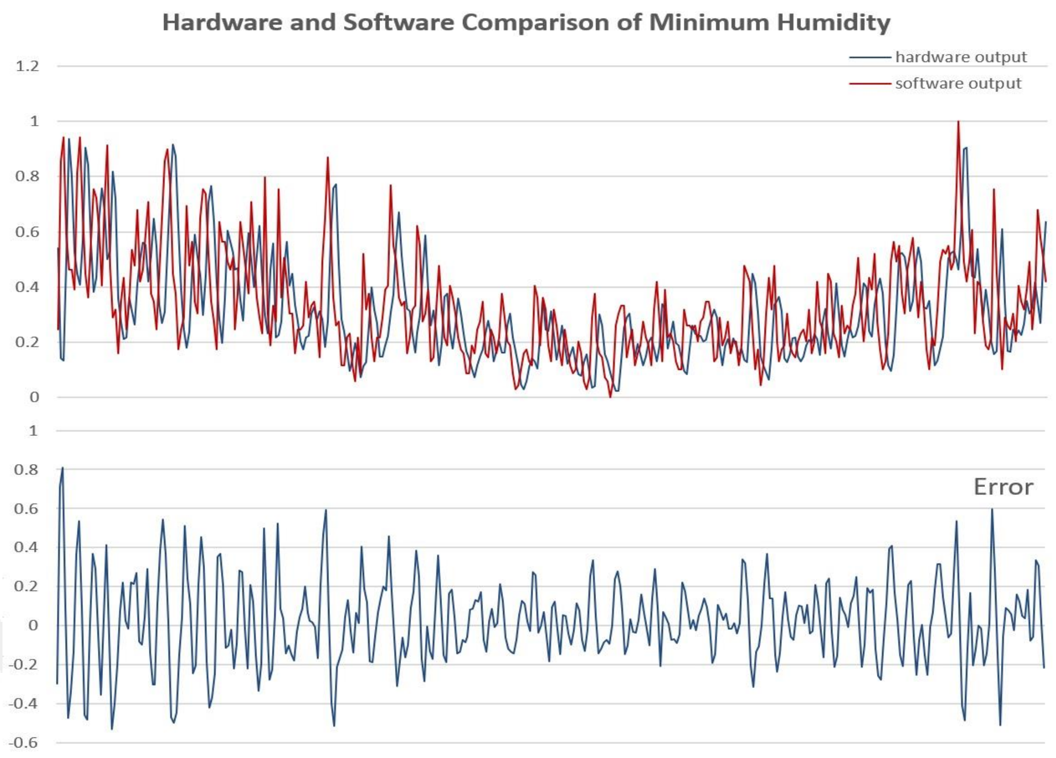


Figure 5.11. Hardware and Software Comparison of Minimum Humidity.

Finally, Table 5.2 lists the comparison values between hardware and software versions in term of the mean of error and the mean square error.

Table 5.2. Comparison between Hardware and Software versions.

Comparison between Hardware and Software versions		
Data set	**MEAN**	**MSE**
Maximum Temperature Set	0.00013	0.0025
Minimum Temperature Set	0.0016	0.0057
Maximum Humidity Set	0.0137	0.0141
Minimum Humidity Set	0.023	0.0459

According to Figures 5.8 to Figures 5.11 and Table 5.2, we can see easily, the hardware version gives very good results in terms of the mean of error and the mean square error and there are very small differences between hardware and software versions in both temperature and humidity data sets. Normally, these small differences appear when using any simulation software compared with hardware systems.

CHAPTER 6
SMART AUTOMATION SYSTEM

6.1. Introduction

In this chapter, hardware design of a smart automation system platform hardware is presented and implemented. The hardware system has two main units: the first unit is The Aggregation unit. It is designed and controlled by Arduino microcontroller that is used as system co-processor. This unit is responsible for real data acquisition from all sensors that are connected to Arduino board, while the second unit is the Multi-function. This unit is designed and implemented based on the Xilinx FPGA Zynq-7 ZC702 evaluation kit. All data analysis tasks like data classification, prediction of data future behaviour, and system decision-making are performed by this unit. At the end of this chapter, and in order to observe and evaluate all information that are received from the hardware system, a special Graphic User Interface (GUI) based on Matlab environment is designed.

6.2. Principle of Smart Automation System

A smart automation system makes the control of various applications and devices in a lot of fields more convenient and quite simple. In general, an automation control system can be utilized for different applications such as weather conditioning, heating, security systems, lighting, and a lot of smart devices.

Mainly, any smart automation system is constructed by actuators, sensors, co-processor unit (microcontroller in most cases), and controlled devices. Different types of real information like light, gas, humidity, temperature, etc. are collected by system sensors. Then, the co-processor sends all sensing data to the main system controller. The co-processor unit permits to bind different actuators and sensors through analog or digital devices. The elements of smart automation system include:

- **Controlled Devices:**

They represent the wide range of smart appliances which can be connected and controlled by co-processor units such as power door locks, door openers, lighting systems, audio and video systems, sprinkling systems, home theater equipment, telephone systems, and other tremendous kinds of equipment.

- **Sensor Devices:**

Sensors devices are used to collect different types of real data such as light, sound, humidity, temperature, etc. The system co-processor mutates these data to another form which can be utilized by the controller to generate appropriate system decisions depend on specific conditions.

- **Interfacing of I/O Devices:**

The interface of I/O devices provide the logical or physical connection links between the system controller and all connected devices in the hardware system. The interface circuits make different devices convenient with the logical and physical architecture of the system. The interface circuit of I/O devices can be isolated or integrated with the system controller. Most of the industrial manufacturers that generate I/O interface circuits utilize standardized protocols which permit products from many manufacturers to interact with each other.

- **System Controllers:**

System Controller implements the intelligent control tasks depend on data which arrive from different sensors. Intelligent tasks which are performed by the system controller, have a wide scope which can be laid from simple to complex tasks. Sufficient data must be provided to the system controller to drive all controlled devices. Providing data can arrive from sensors, timers, user, control program, or any combination of them. Figure 6.1 presents the main parts of an intelligent control system.

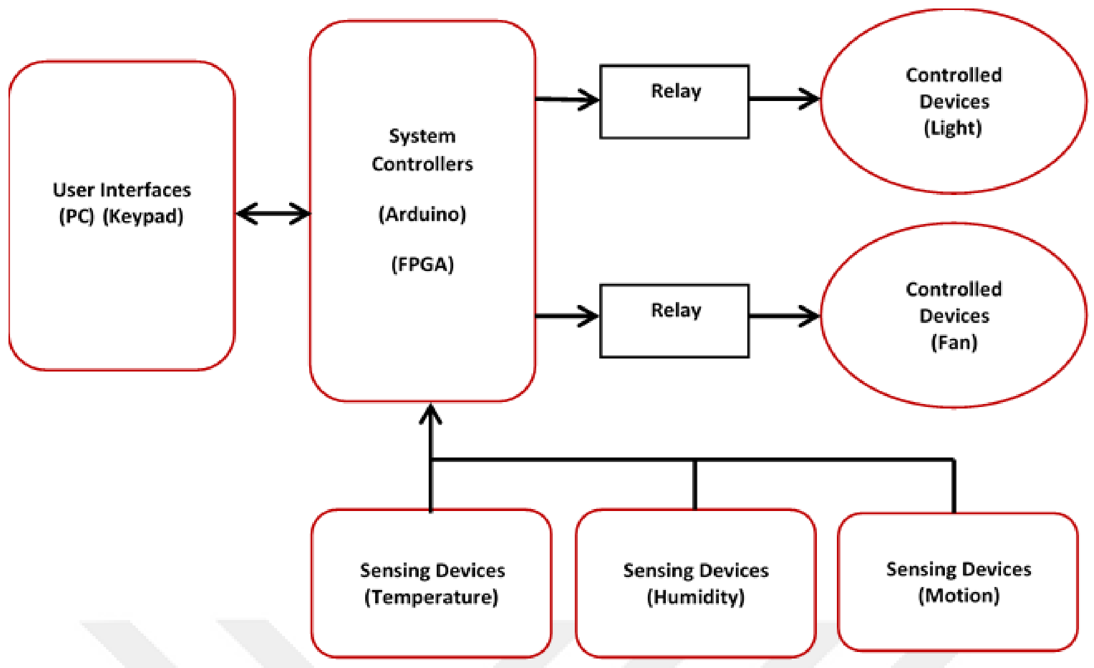


Figure 6.1. Main parts of Smart Automation System.

A special and simple graphic user interface is used to react the user with the system controller and monitor all arrived information or send user commands to the system about certain conditions. However, in some advanced automation systems supply intelligent decisions without the need to any graphical user interfaces.

Power consumption, response time, cost, and flexibility are most challenges that any Automation Systems face. So, combining Artificial Intelligence (AI) systems with embedded systems based on programmable devices such as FPGA automation control system can face all the above challenges.

FPGAs provide a significant processing system, depend on application-specific integrated circuits. FPGA devices have powerful mathematical and rapid acceleration components that qualify these devices to perform simple and more complex algorithms.

6.3. Designing of Automation Control System

The proposed hardware system aims to provide an automation control system that is flexible, high-efficient and fast. The proposed system structure mainly consists of three units:

1) **Aggregation Unit:** This unit is implemented based on Arduino microcontroller and acts as a system co-processor. The unit performs the data aggregation task from different analog and digital sensors

2) **Multi-function Unit**: Multi-function Unit is designed and implemented by Xilinx FPGA evaluation board and acts as the main system controller.

3) **Communication Unit**: It is used to connect the Multi-function unit, the Aggregation Unit, and Personal Computer (PC).

6.3.1. Aggregation Unit

As we said before, the aggregation data from different sensors are done by this unit. So, 3-sets of different sensors are connected to this unit based on the Arduino Uno microcontroller. Each set consists of:

- 3-analog gas sensors
- 3-digital temperature sensors and,
- 3-digital humidity sensors

The microcontroller collects the data from all these sensors and associates each byte of sensing data with a special 8-bit ID. After that, the Arduino re-arrange all data and generates the Ethernet message format in order to forward data to the Multi-function unit. Figure 6.2 demonstrates the Ethernet message between Multi-function and Aggregation units.

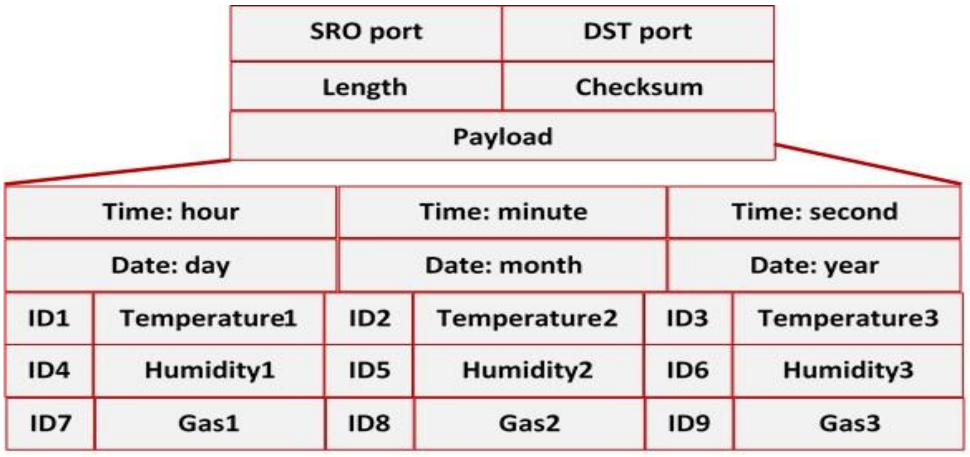


Figure 6.2. Ethernet Message between Multi-function and Aggregation.

According to the Figure 6.2, we used 27-bytes in payload field within the Ethernet message format. The payload can be divided into three that are distributed on three groups of sensors in form of:

- 8-bit ID / 8-bit temperature sensor values,
- 8-bit ID / 8-bit humidity values, and
- 8-bit ID / 16-bit gas sensor values.

A 6-byte is used in the beginning to provide system time and date in the form of:

- hour: minute: second for system time and,
- day: month: year for date.

6.3.2. Multi-function unit

The Multi-function unit is the second unit in our hardware design. This unit is designed based on Vivado HLS V2016.1, Vivado Design Suite V2016.1, and Vivado SDK V2016 and targeted at the Xilinx Zynq-7 ZC702 evaluation kit. The Multi-function unit is obtained and the sensing data from the Aggregation Unit and starts to perform multiple tasks and data analysis on the receiving data. These tasks include:

- Data Classification,
- Prediction of future values based on Modified ANFIS Algorithm,
- System Decision-making.
- System Fault Alerts.

Figure 6.3 shows the Multi-function unit based on Vivado Design Suite.

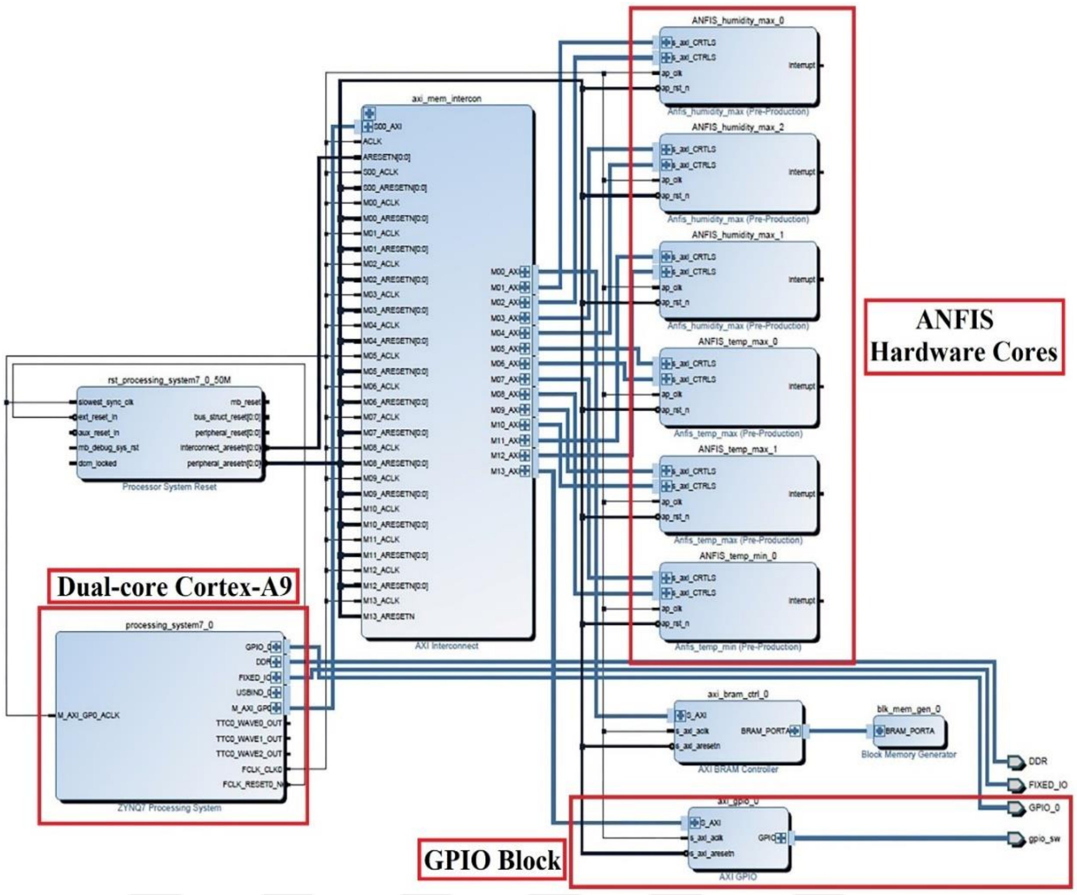


Figure 6.3. Hardware Design within the Multi-Function.

Three main parts are included in the Figure 6.3 and denoted by the labels as shown below:

- **The Dual-core SYNQ Cortex-A9 Processor**: This is the main system controller. The initialization of hardware platform, implements and controls all tasks within this unit are done by this processor which acts as a main system controller. The flowchart in Figure 6.4 explains all tasks that are executed by SYNQ processor.

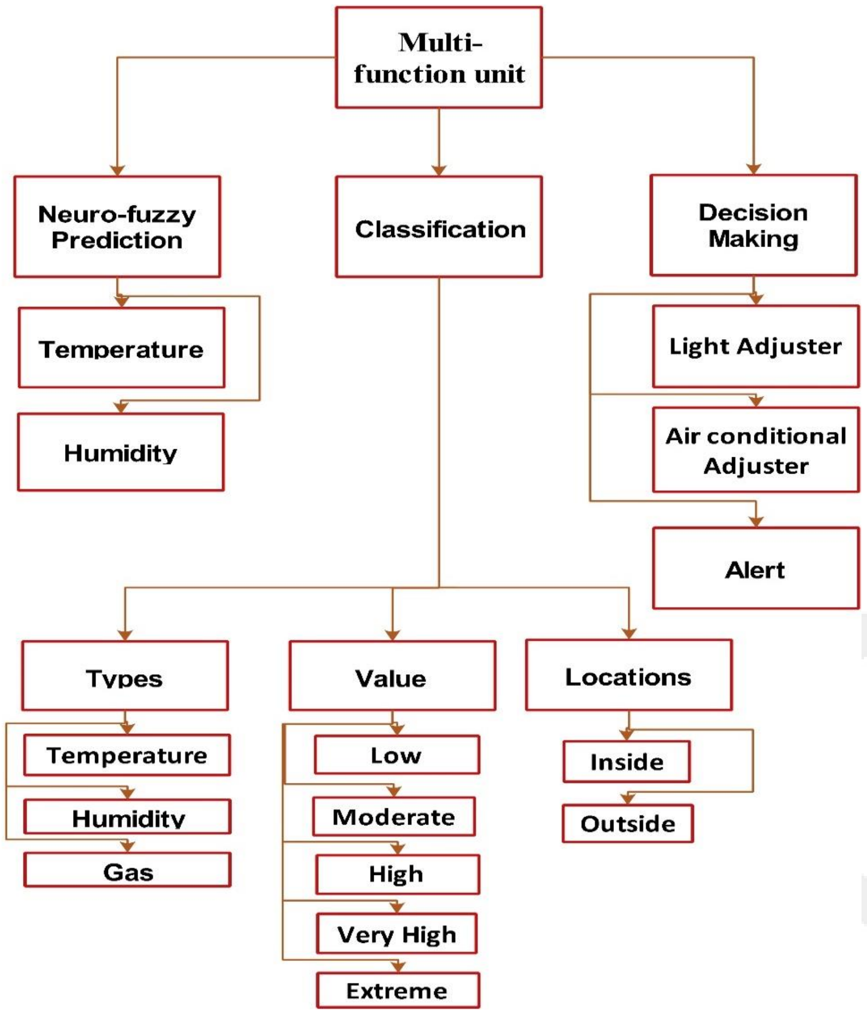


Figure 6.4. The overall tasks are executed by Cortex-A9.

The neuro-fuzzy prediction task in the above flowchart is implemented by the modified ANFIS algorithm based on time series approach for both temperature and humidity data. While the classification task is done according to:

1. Types (gas sensors, humidity, and temperature),
2. Values (extreme, very high, high, moderate, and low), and
3. Locations (outside or inside the home).

The last task is system decision making. This task is done based on:

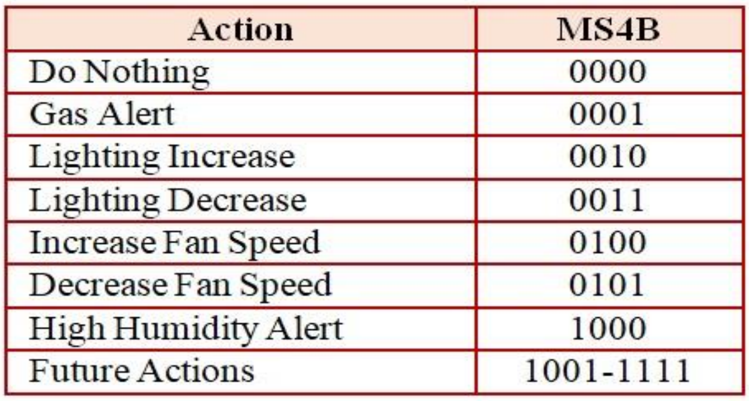
1. Light adjuster,
2. Air conditional adjustment, and
3. System Alert.

- **ANFIS Hardware Cores:** As we said in the previous section, our system design includes 3-humidity and 3-temperature sensors linked to the aggregation unit and the prediction processes are performed on the data coming from these sensors. The prediction results are sent back to the system processor to prevent any unusual event in the automation control system. For this reason, the designing system has 6-ANFIS cores. Each core is allocated to the particular sensor.

- **General Purpose I/O Ports (GPIO):** GPIO is the last block in the Multi-function unit. The main processor will evaluate all data (original data and results) in order to generate the final decisions and the system fault alerts (4-bit binary code is used to represent each one). The processor will send those codes to the GPIO block.

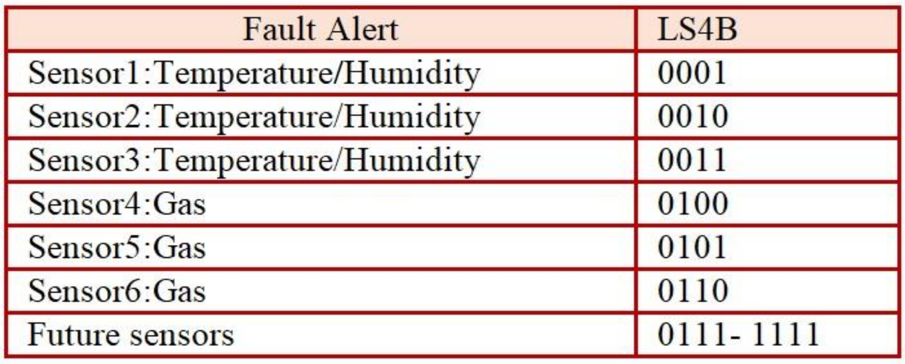
Table 6.1 and Table 6.2 explain how can represent the system fault alerts and decisions by 8-bit binary codes respectively.

Table 6.1. System Actions.



Action	MS4B
Do Nothing	0000
Gas Alert	0001
Lighting Increase	0010
Lighting Decrease	0011
Increase Fan Speed	0100
Decrease Fan Speed	0101
High Humidity Alert	1000
Future Actions	1001-1111

Table 6.2. System Fault Alerts.



Fault Alert	LS4B
Sensor1:Temperature/Humidity	0001
Sensor2:Temperature/Humidity	0010
Sensor3:Temperature/Humidity	0011
Sensor4:Gas	0100
Sensor5:Gas	0101
Sensor6:Gas	0110
Future sensors	0111- 1111

The generated 8-bit binary code is divided into two parts: the codes in Table 6.1 represent the Most Significant 4-bit (MS4B) which is used for the system actions, while the codes in Table 6.2 represent the Least Significant 4-bit (LS4B) for the system fault alerts.

The Least Significant 4-bit (LS4B) in Table 6.1 are forwarded to the J62 headers in the Xilinx Zynq-7 ZC702 evaluation kit. The system processor sends the system actions commands as a 4-bit to the J62 header.

Figure 6.5 illustrates the pseudo code of the system actions based on 4-bit commands that can be achieved.

```
Initialization of maximum threshold temperature value
Initialization of minimum threshold temperature value
Initialization of maximum threshold humidity value
Initialization of minimum threshold humidity value
 Initialization of gas threshold value
Initialization of time threshold value
Recieve the data from Arduino

// start data processing
while (data is available) then

// deal with temperature sensors
for (each temperature sensor) do
      if (temperature value > maximum threshold) then
     Increase Fan Speed: Code=0100
     if (temperature value < minimum threshold) then
    Decrease Fan Speed: Code=0100
end for loop


 //deal with humidity sensors
for (each humidity sensor) do
     if (humidity value > threshold) then
     High Humidity Alert: Code=1000
end for loop

 // deal with gas sensors
for (each gas sensor) do
      if (gas value > threshold) then
      Gas Alert: Code=0001
end for loop
```

```
                         // Continue //
// deal with lighting
If (time: hour > threshold) then
      Lighting: Turn-On: Code=0010
     Lighting: Turn-Off: Code=0011

end while loop

end while loop

De-normalize the output data
```

Figure 6.5. Pseudo Code of the System Actions.

While Figure 6.6 shows 4-LED interfacing to the J62 header in FPGA board.

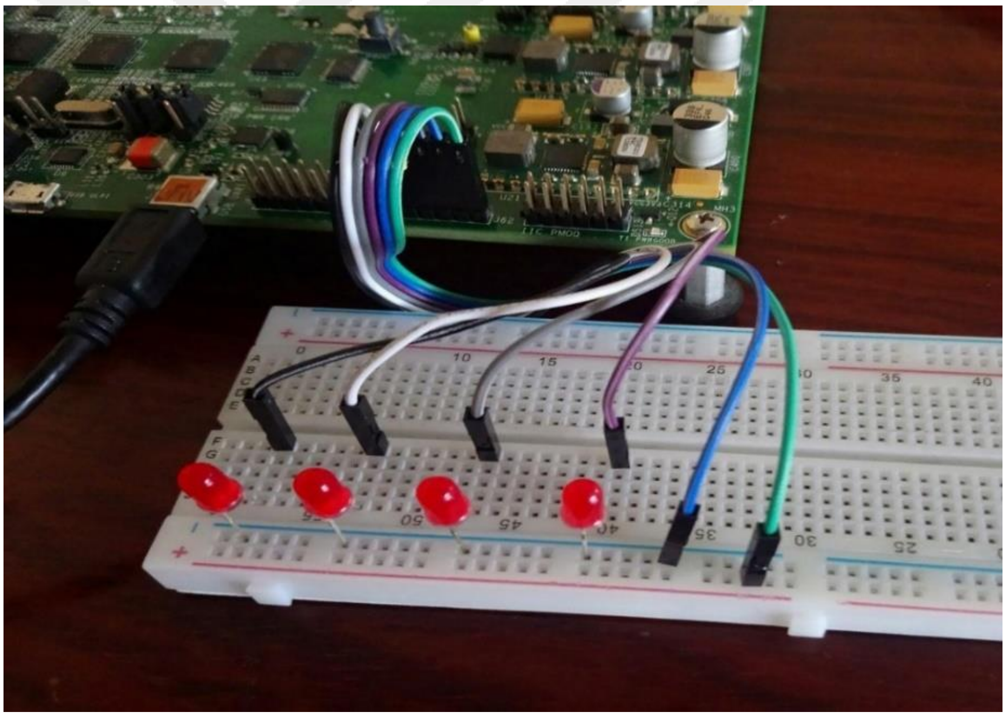


Figure 6.6. The Interface of the J62 Header.

6.3.3. Communication Unit

The Communication Unit is the last unit in our hardware design. Basically, this unit is implemented by a special library known as Lightweight IP (lwIP) that is used by Gigabit Ethernet Controllers (GEM) within Zynq-7 ZC702 board. The lwIP library is a TCP/IP networking stack and an open source that is used for embedded systems. The

Xilinx (SDK) provides and customizes this library to work on different Xilinx embedded system environments such as Xilinx ARM-based Zynq-7000, PowerPC hard processor, or soft MicroBlaze-based processor (Xilinx, 2016). The lwIP library can perform the following protocols(Macmahon & Cherukupaly, 2014):

- Internet Protocol (IP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Address Resolution Protocol (ARP)
- Dynamic Host Configuration Protocol (DHCP)
- Internet Group Message Protocol (IGMP)
- Internet Control Message Protocol (ICMP)

The lwIP library helps to provide networking capability to the FPGA embedded system and minimize the number of resources that are used to achieve a network TCP stack. Now, the main goal of the Communication Unit is to connect the multi-function unit to the aggregation unit and forward data to the user interface panel.

To achieve this goal, we develop a new version of Lightweight IP (lwIP) library that supports multi-server and multi-client devices by combining between a standard client version and a standard server version in a single library-based FPGA device that ready to use by researchers in IoT applications. The newly developed library is modified based on User Datagram Protocol (UDP).

7.3.3.1. Xilinx Adapter Functions

The Xilinx adapters-based SDK tool provides multiple functions in order to simplify the implementation of the lwIP Application Programming Interfaces (lwIP APIs) that are used to modify the Communication Unit. The lwIP library provides two different APIs: RAW mode and Socket mode. In our thesis, we will focus on RAW mode which is callback based. This mode provides a perfect performance at the price of compatibility compared with other TCP networking stacks. In this mode, API gets direct access into the TCP stack and there is no needed to additional socket layer (Xilinx, 2016; Macmahon and Cherukupaly, 2014) as explained below:

- void **lwip_init**()

This function provides the initialization of the lwIP functions for the data structures. This function replaces specific calls to initialize system memory, pbufs, UDP, IP, and TCP layers. Figure 6.7 shows the callback of this function.

```
struct netif *xemac_add (struct netif *netif, struct
    ip_addr *ipaddr, struct ip_addr *netmask, struct
    ip_addr *gw, unsigned char *mac_ethernet_address
    unsigned mac_baseaddr)
```

Figure 6.7. Initialization of the Lwip Functions.

The **xemac_add** function supplies a unified interface for Xilinx EMAC IP.This function covers the lwIP: network interface (*netif_add*), IP address (*ipaddr*), gateway IP address (*gw*), ethernet address MAC address(*mac_ethernet_address*), and memory base address of the MAC address(*mac_baseaddr*).

- void **xemacif_input**(struct netif **netif*)

This function is used to handle the system interrupt in order to put the arrived data packets in a queue, while the xemacif_input function moves those packets from the queue, and presents them to lwIP. Figure 6.8 shows this process.

```
while (1)  {
               /* receive packets */
               xemacif_input(netif);

               /* do application specific processing */
       }
```

Figure 6.8. handling the system interrupt by lwIP library.

The pseudocode in Figure 6.9 shows the initialization structure of the RAW API that we are used in this thesis.

```
int main()
{
        struct netif *netif, server_netif;
        struct ip_addr ipaddr, netmask, gw;

        /* the MAC address of the board.
        * This should be unique per board/PHY */
        unsigned char mac_ethernet_address[] =
            {0x00, 0x0a, 0x35, 0x00, 0x01, 0x02};

        lwip_init();

        /* Add network interface to the netif_list,
         * and set it as default */
        if (!xemac_add(netif, &ipaddr, &netmask,
            &gw, mac_ethernet_address,
            EMAC_BASEADDR)) {
            printf("Error adding N/W interface\n\r");
            return -1;
        }
        netif_set_default(netif);

        /* now enable interrupts */

        platform_enable_interrupts();

        /* specify that the network if is up */
        netif_set_up(netif);

        /* start the application, setup callbacks */
        start_application();

        /* receive and process packets */
        while (1) {
            xemacif_input(netif);
            /* application specific functionality */
            transfer_data();
        }
 }
```

Figure 6.9. Initialization Structure of the RAW API.

Several sequence functions are used to establish the connection between the server and client devices based on this library. These functions are used to create a TCP connection as shown in Figure 6.10.

```
/* create new TCP PCB structure */
pcb = tcp_new();

/* bind to specified @port */
err = tcp_bind(pcb, IP_ADDR_ANY, port);

/* we do not need any arguments to callback functions */
tcp_arg(pcb, NULL);

/* listen for connections */
pcb = tcp_listen(pcb);

/* specify callback to use for incoming connections */
tcp_accept(pcb, accept_callback);
```

Figure 6.10. Establishing the Connection based on Lwip.

The accept_callback function is called automatically when the connection is accepted, because the server device needs to reply only when data arrives. At this time, the function recv_callback is called in order to send the data to the other device as explained in Figure 6.11.

```
/* set the receive callback for this connection */
tcp_recv(newpcb, recv_callback);

/* indicate that the packet has been received */
tcp_recved(tpcb, p->len);

/*  send data to the other device
err = tcp_write(tpcb, p->payload, p->len, 1);
```

Figure 6.11. Data Arrived and Received Callbacks.

The flowchart in Figure 6.12 shows all functions that are required to establish the mult-server and multi-client based on Lwip.

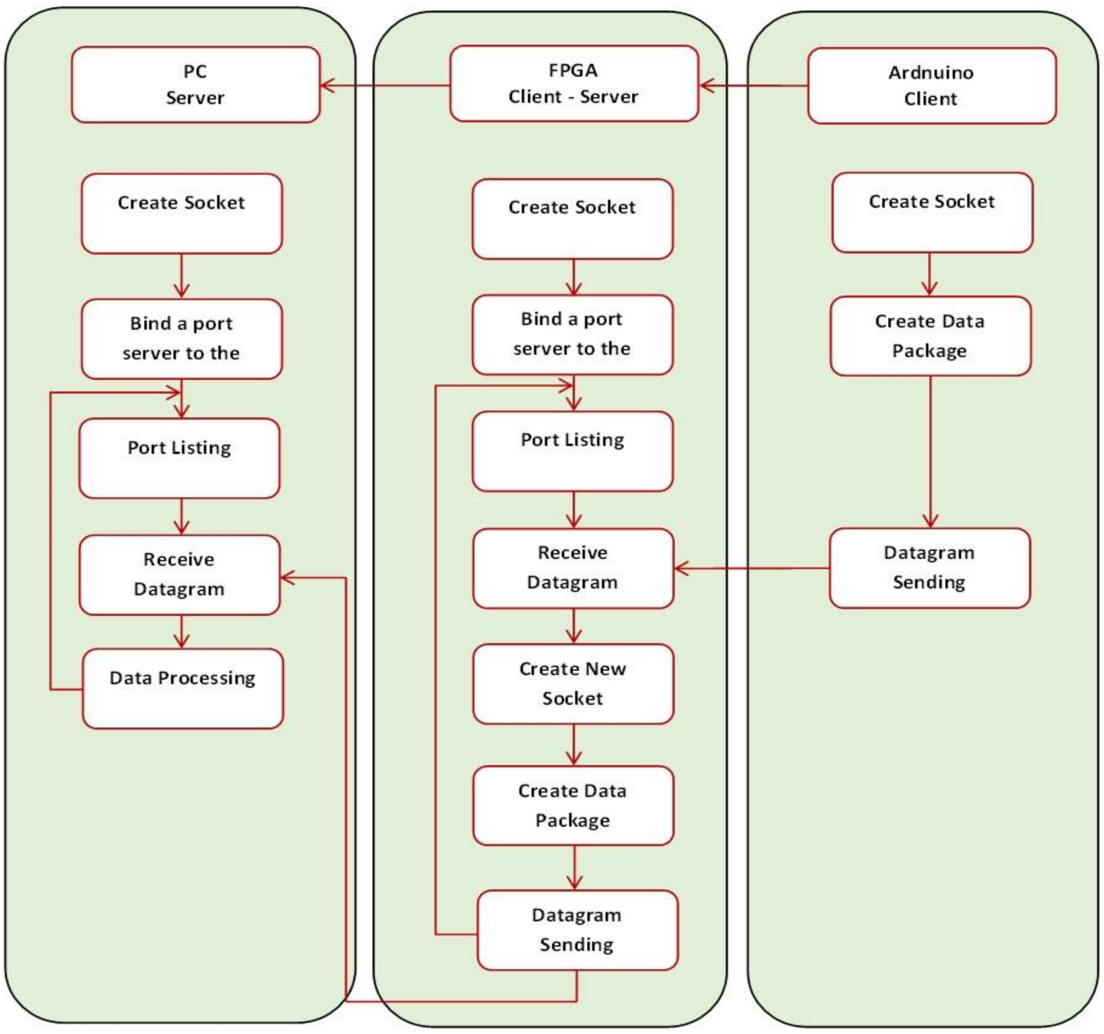


Figure 6.12. Modified LWIP based on User Datagram Protocol.

According to Figure 6.12, the Dual-core SYNQ Cortex-A9 Processor begins as system server which continuously listens to received sensing data from the different sensors in the Aggregation Unit. Then, the Processor changes its status from server to UDP client to send all data to the user interface panel-based Matlab Graphic User Interface (GUI) after processes it.

Finally, Figure 6.13 shows the complete proposed system design that includes all system units.

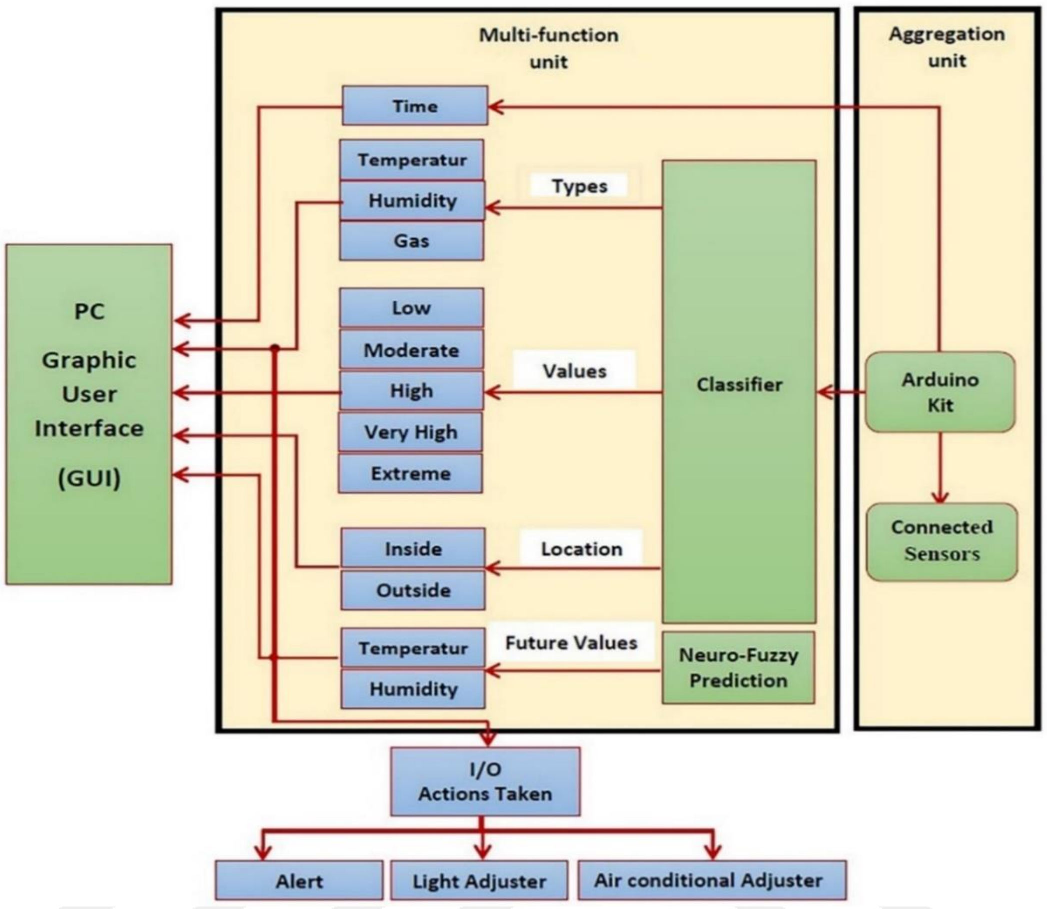


Figure 6.13. Complete Proposed Hardware System.

6.4. Experimental Results

The overall hardware design is tested and evaluated during all implementation levels to validate the performance of the system. The data acquisition- based Arduino Uno kit is done every 15-minute and then the Arduino microcontroller forwards the sensing data to the Zynq-7 ZC702 board through the Ethernet cable. Figure 6.14 presents the Aggregation unit and how the system’s sensors are connected to the Arduino Uno board. According to Figure 6.14**,** it includes:

- Arduino Uno kit.
- DS3231 Real Time Clock (RTC) module which supplies a time system.
- 3-MQ2 Gas Sensors.
- Arduino Ethernet Shield-2 is utilized to provide the network capability to the Arduino Uno controller in order to connect the Arduino Uno board (Aggregation Unit) to the FPGA evaluation kit (Multi-function unit).

- 3-DHT11 Temperature
- 3-DHT11 Humidity sensors.

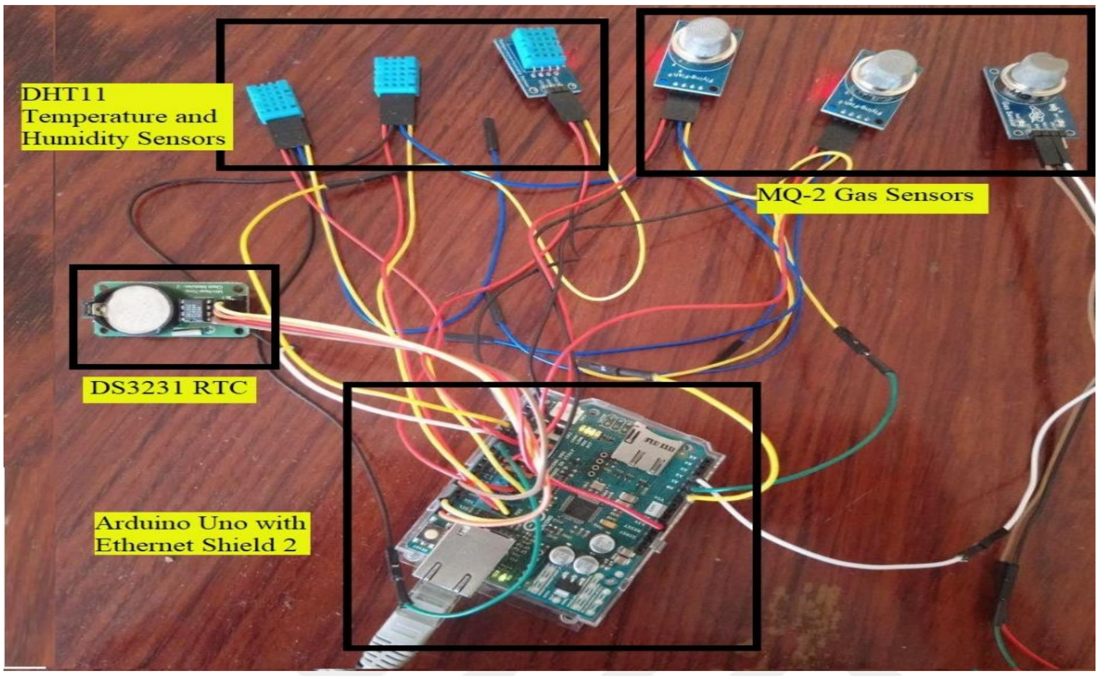

DHT11 Temperature and Humidity Sensors
MQ-2 Gas Sensors
DS3231 RTC
Arduino Uno with Ethernet Shield 2

Figure 6.14. Interfacing Arduino with the Sensors.

Figure 6.15 shows how the Arduino board connects to Xilinx Zynq-7 ZC702 evaluation kit through the Ethernet cable.

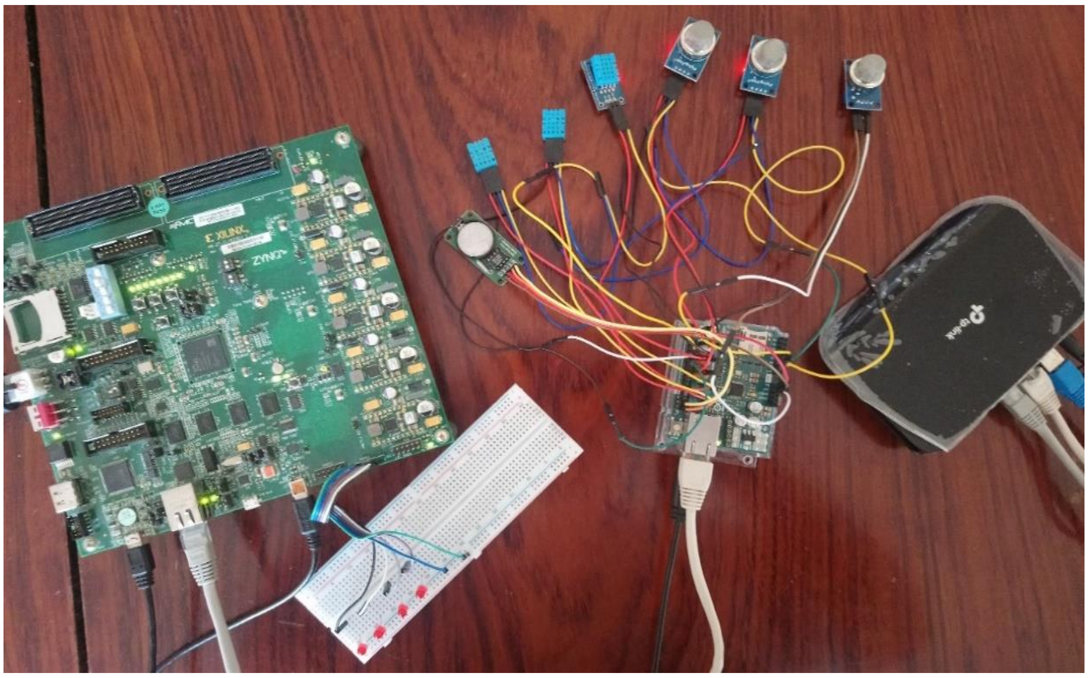
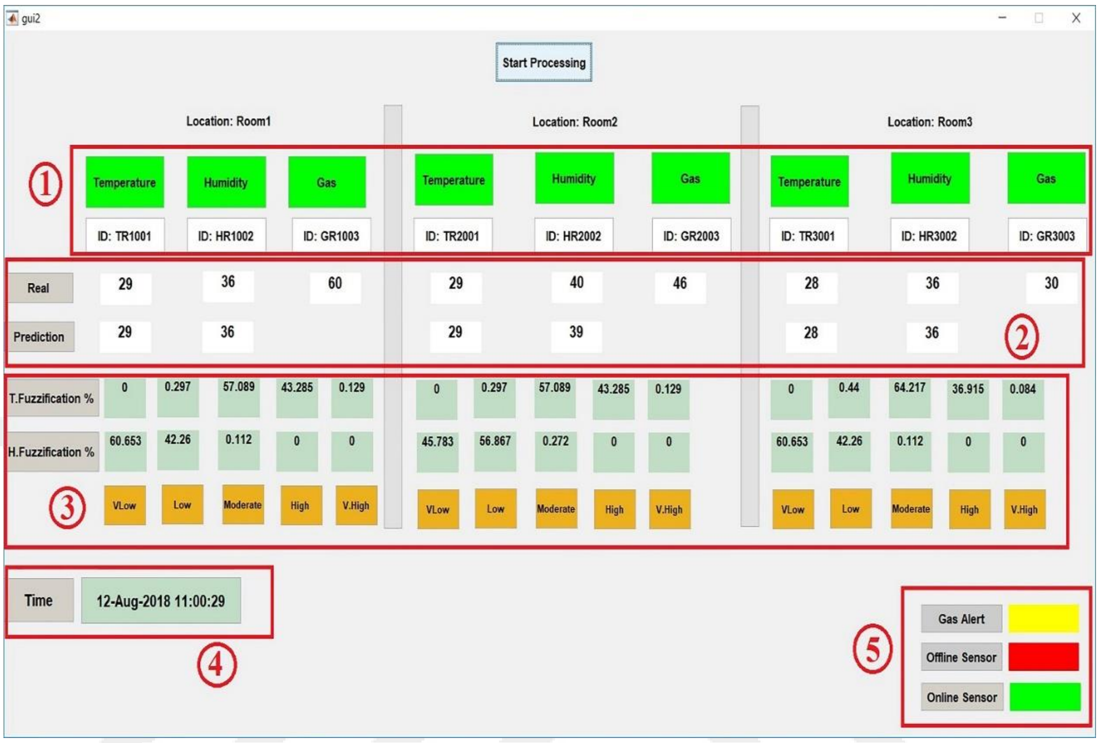


Figure 6.15. The Connection between Arduino Board and FPGA.

To monitor and evaluate information that arrives from hardware system units, a Matlab GUI that acts as a user panel is built for this purpose. based on the environment is designed. The user panel is demonstrated in Figure 6.16.


gui2

Start Processing

	Location: Room1			Location: Room2			Location: Room3		
1	Temperature	Humidity	Gas	Temperature	Humidity	Gas	Temperature	Humidity	Gas
	ID: TR1001	ID: HR1002	ID: GR1003	ID: TR2001	ID: HR2002	ID: GR2003	ID: TR3001	ID: HR3002	ID: GR3003
Real	29	36	60	29	40	46	28	36	30
Prediction	29	36		29	39		28	36	2

T.Fuzzification %	0	0.297	57.089	43.285	0.129	0	0.297	57.089	43.285	0.129	0	0.44	64.217	36.915	0.084
H.Fuzzification %	60.653	42.26	0.112	0	0	45.783	56.867	0.272	0	0	60.653	42.26	0.112	0	0
3	VLow	Low	Moderate	High	V.High	VLow	Low	Moderate	High	V.High	VLow	Low	Moderate	High	V.High

Time 12-Aug-2018 11:00:29

4

5

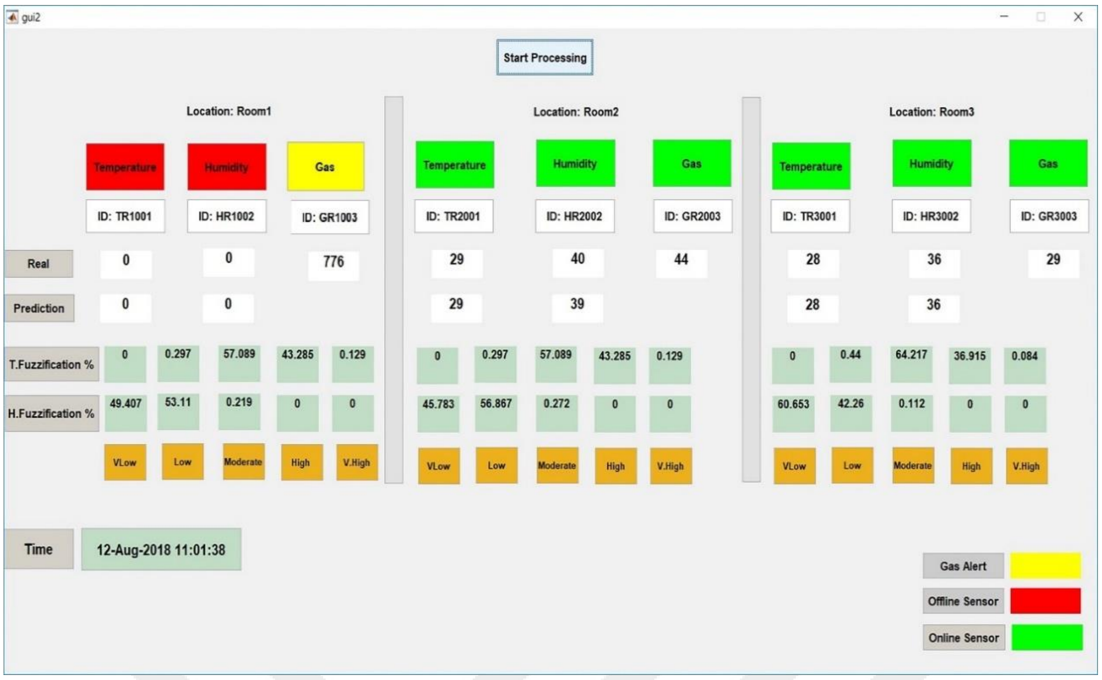
Gas Alert	🟨
Offline Sensor	🟥
Online Sensor	🟩

Figure 6.16. System Graphic User Interface.

Figure 6.11 includes five main parts which are highlighted and numbered by the red-color as illustrated below:

- **Part 1:** This part is used to display the connected sensors and its ID.
- **Part 2:** This section is used to monitor all sensing data that come from sensors and the prediction of future values of humidity and temperature.
- **Part 3:** Classification of each group of data according to its type and value and provide the percentage for each value is done in this part.
- **Part 4:** The information of date and time (real-time clock) is displayed here.
- **Part 5:** The last part is used to give information about the system alerts colors. The green color is utilized for on-line sensors, while yellow color is utilized for gas detector alert. Finally, the red color is used to alert for off-line sensors.

Figure 6.17 shows an example of high-level gas detector and humidity/temperature sensors in the off-line state (disconnected).


gui2

Start Processing

	Location: Room1			Location: Room2			Location: Room3		
	🟥 Temperature	🟥 Humidity	🟨 Gas	🟩 Temperature	🟩 Humidity	🟩 Gas	🟩 Temperature	🟩 Humidity	🟩 Gas
	ID: TR1001	ID: HR1002	ID: GR1003	ID: TR2001	ID: HR2002	ID: GR2003	ID: TR3001	ID: HR3002	ID: GR3003
Real	0	0	776	29	40	44	28	36	29
Prediction	0	0		29	39		28	36	

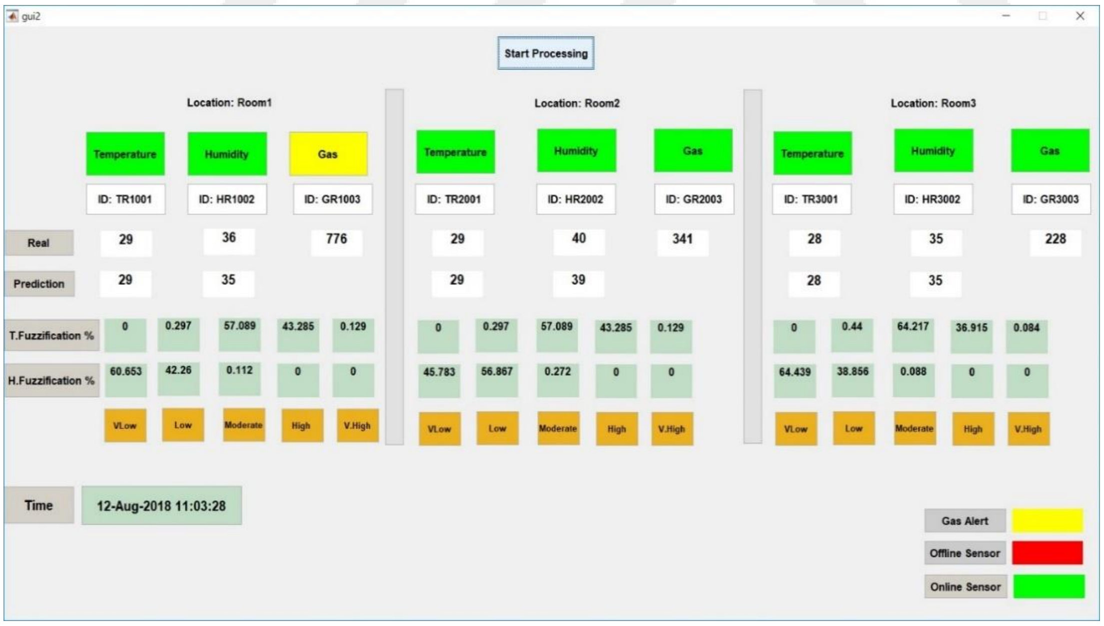
	VLow	Low	Moderate	High	V.High	VLow	Low	Moderate	High	V.High	VLow	Low	Moderate	High	V.High
T.Fuzzification %	0	0.297	57.089	43.285	0.129	0	0.297	57.089	43.285	0.129	0	0.44	64.217	36.915	0.084
H.Fuzzification %	49.407	53.11	0.219	0	0	45.783	56.867	0.272	0	0	60.653	42.26	0.112	0	0

Time 12-Aug-2018 11:01:38

Gas Alert 🟨
Offline Sensor 🟥
Online Sensor 🟩

Figure 6.17. High-level Gas Detector and Temperature/Humidity in the Off-line State.

Other example is presented in Figure 6.18 when a high-level gas is detected while other sensors work properly.


gui2

Start Processing

	Location: Room1			Location: Room2			Location: Room3		
	🟩 Temperature	🟩 Humidity	🟨 Gas	🟩 Temperature	🟩 Humidity	🟩 Gas	🟩 Temperature	🟩 Humidity	🟩 Gas
	ID: TR1001	ID: HR1002	ID: GR1003	ID: TR2001	ID: HR2002	ID: GR2003	ID: TR3001	ID: HR3002	ID: GR3003
Real	29	36	776	29	40	341	28	35	228
Prediction	29	35		29	39		28	35	

	VLow	Low	Moderate	High	V.High	VLow	Low	Moderate	High	V.High	VLow	Low	Moderate	High	V.High
T.Fuzzification %	0	0.297	57.089	43.285	0.129	0	0.297	57.089	43.285	0.129	0	0.44	64.217	36.915	0.084
H.Fuzzification %	60.653	42.26	0.112	0	0	45.783	56.867	0.272	0	0	64.439	38.856	0.088	0	0

Time 12-Aug-2018 11:03:28

Gas Alert 🟨
Offline Sensor 🟥
Online Sensor 🟩

Figure 6.18. An Example of High-level Detection.

Compared with existing implementation of automation control systems that related to our approach, we can divide them into two group:

1. The hardware implementations in the first group are designed and implemented systems are based on traditional microcontrollers(Kumar, Shegokar, Chouhan, and Iqbal, 2018; Asadullah and Raza, 2016) and AI (Artificial Intelligent) algorithms the designers are integrated with these systems in order perform data analysis and smart self-decision-making like some commercial systems such as Alexa from Amazonis and Apple HomeKit and several academic works like (Prabhu, Jena, and Rode, 2018;; Sang-hyun et al., 2013; Yugashini, Vidhyasri, and Devi, 2013).

2. In the second group, the designers use the FPGA devices in their design of automation systems in order to take advantage of their flexibility and reconfigurability features like implementations in (Suresh and Mastani, 2018; Panigrahi et al., 2016; Chinchansure and Kulkarni, 2014; D, 2017; Assaf et al., 2012; Pratik Waghmare, Chandgude, Chaure, and Chaudhari, 2017).

To build a robust, fast, and high performance, our proposed implementation combines between the above two groups. The system is designed and implemented in reconfigurability FPGA environment and take advantage of its flexibility. Also, our system has multiple Artificial Intelligent algorithms that can perform simple and complex tasks based on its application.

CHAPTER 7
CONCLUSION AND FUTURE WORKS

7.1. Conclusion

In order to enhance sensor nodes architecture and its computation power, Field-Programmable Gate Array (FPGA) is used to design a modified hardware structure of a sensor node. This structure is embedded with an ensemble of the Adaptive Neuro-Fuzzy Inference system algorithms that is one of the artificial intelligence algorithms to present a typical solution of the limitations of sensor node-based classical microcontrollers. The contribution of this thesis can be divided into three parts:

1) **In the first part**, the Adaptive Neuro-fuzzy Inference System has been used, modified and evaluated based on different data sets. Two modifications are added to the ANFIS algorithm in order to enhance its work and performance (Chapter 4):

    - During the first modification, the ANFIS algorithm is combined with a momentum factor which is a type of optimization algorithms during the learning process. The momentum factor helped the ANFIS algorithm to arrive at the optimal solution and provided a best and fast convergence without stuck in local minima and reduce the overall error compared with the standard architecture of ANFIS algorithm. in terms of Mean Squared Error, Root-Mean-Square Error, Mean Error, and Standard Deviation. Using momentum factor in our ANFIS algorithm minimizes the execution time (in second) during the training phase from 3.675 to 1.1921 and from 3.6427 to 1.1895 for temperature and humidity data sets respectively that are used to test the first modification.

- **In the second modification**, we proposed a methodology that reduces the adjustable linear parameters at layer 4 of ANFIS algorithm. This process is done by adding a special normalization that is used compress linear equations at the output layer without effect on the overall performance of the algorithm.

Several different data sets related to weather condition are used to test and evaluated this modification. All results are compared with ANFIS algorithms-based Matlab toolbox. High accuracy is achieved in both training and testing phases based on the modified algorithm compared with the ANFIS toolbox (Section 4.4.1.2).

Minimizing the number of linear parameters do not affect the overall performance of the modified algorithm. Also, the execution time of the training and testing phases are measured (Figure 4.29 and Figure 4.30). Adding this modification to the algorithm provides high speed in the training algorithm compared with ANFIS-based Matlab tool and good speed in the testing algorithm.

2) **In the second part of our contribution** is the design and implementation of the proposed hardware system that is done and evaluated in Chapter 5. The achievements can be divided into:

- **An ensemble of 6-soft hardware cores-based modified ANFIS algorithm is built and developed using the reconfigurable FPGA Zynq xc7z020 evaluation kit.** The ensemble of ANFIS cores is designed to work in parallel and deal with different real acquisition data. Each ANFIS core is built based on three associated tools:

  ❖ Xilinx Vivado High-Level Synthesis V2016.1 that is used to build 6-hardware cores of modified ANFIS-based time series algorithm. A small C programming language is used in this tool which is suitable for programming the algorithms in embedded systems.

  ❖ Xilinx Vivado Suite V2016.1 that is used to build the complete hardware design and import 6-ANFIS cores from Vivado High-Level Synthesis libraries. Then, export all hardware files and libraries to the Xilinx Software Development Kit.

❖ Xilinx Software Development Kit tool which is used to initialize ARM Cortex-A9 processor and build the software platform that includes drivers, system parameters, hardware core and other associated hardware elements.

- **The ARM Cortex-A9 processor is used to share the primary tasks of ANFIS algorithm (data normalization and data fuzzification tasks) with ANFIS-hardware cores.** In the other word, we successfully distributed the processing tasks between Zynq programming logic (PL) which is represented by ANFIS-cores and Zynq processing system (PS) which is represented by ARM Cortex-A9 processor (Section 5.3). This helps us to minimize the DSP block by 55%.

- **Real sensing data based on 3-temperature and 3-humidity DHT11 sensors that grouped in 3-set are used to test the performance of the hardware system.** Compared the hardware results-based ANFIS cores with the simulation results-based Matlab software provide very good results in terms of the mean of error and the mean square error (Section 5.4) with very small error ratio which is the normal case when comparing hardware with software results.

3) **The last part of our contribution** is the design of monitoring and intelligent control system**.** The system consists of several parts:

- **Aggregation Unit-based Arduino microcontroller** is used to connect 3-set of sets of temperature, humidity, and gas sensors. We used the Arduino microcontroller as a co-processor in our system besides the ARM Cortex-A9 processor in order to aggregate and send the sensing data to the next unit in the monitoring system.

- **Multi-function unit is constructed based on Zynq-7 ZC702 kit**. This unit received the data from the Aggregation Unit and performs:
  - ❖ Data classification,
  - ❖ Prediction future behavior of the weather condition,
  - ❖ System decision-making, and
  - ❖ System Alerts.

- **Communication Unit based on Lightweight IP library** that is used especially for embedded systems to provide them with a networking capability (Section 6.3.3). The aim of this unit is to connect the system units with each other (multi-function unit, aggregation unit and user panel). We modified a special multi-server and multi-client agents based on the User Datagram Protocol (UDP). The main purpose of this modification is to allow the multi-function unit to switch its mode based on system status.

- User panel which is designed based on Matlab Graphic User Interface to monitor all data that is come from the hardware system.

Using a combination of Arduino microcontroller and FPGA device plays a key role to distribute the tasks between microcontroller that interfaces different types of sensors and FPGA that performs the data analysis and pre-processing (François Philipp, 2014). As a result, we reduced the execution time, the required hardware resources and add added more flexibility to the system to implement more complex and intelligent functions.

Comparing our system with the existing implementation of automation control systems (Section 6.4), our proposed implementation combines the traditional control system that has embedded Artificial Intelligent algorithms (to perform data analysis and smart self-decision-making) and other systems that are built by FPGA environments. Our system is implemented in FPGA environment in order to take advantage of its flexibility and reconfigurability. Also, the system has multiple Artificial Intelligent algorithms that are utilized to implement simple and complex tasks based on its application.

7.2. Future Works

This research aims to build a smart and reconfigurable sensor node based on FPGA device. But, the continuation of such a contribution is open. The overall system consists of single Aggregation Unit and Multi-function unit that connect together by ethernet cable based on UDP protocol. So, update the system can include the following:

- Using multiple Aggregation Unit and modifying the communication protocol to support multiple agents wirelessly.

- Adding security algorithms to the system in order to protect the system information.

- Adding the partial reconfigurable feature to the system to change or modify some functions in the system (or functions in some agent in case of multiple agents).

References:

Abraham, A. (2001). Neuro fuzzy systems: State-of-the-art modeling techniques. In *International Work-Conference on Artificial Neural Networks* (pp. 269–276). Springer.

Aliev, R. A., & Guirimov, B. G. (2014). *Type-2 fuzzy neural networks and their applications*. Springer.

Altera. (2010). Stratix V Device Family Overview. *Stratix V Device Handbook, 1*(March), 1–15.

Amari, S. (2003). *The handbook of brain theory and neural networks*. MIT press.

Anderson, D., & McNeill, G. (1992). Artificial neural networks technology. *Kaman Sciences Corporation, 258*(6), 1–83.

Asadullah, M., & Raza, A. (2016). An overview of home automation systems. *2016 2nd International Conference on Robotics and Artificial Intelligence, ICRAI 2016*, 27–31. https://doi.org/10.1109/ICRAI.2016.7791223

Assaf, M. H., Mootoo, R., Das, S. R., Petriu, E. M., Groza, V., & Biswas, S. (2012). Sensor based home automation and security system. *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, 722–727. https://doi.org/10.1109/I2MTC.2012.6229153

Bellis, S. J., Delaney, K., O'Flynn, B., Barton, J., Razeeb, K. M., & O'Mathuna, C. (2005). Development of field programmable modular wireless sensor network nodes for ambient systems. *Computer Communications*, *28*(13 SPEC. ISS.), 1531–1544. https://doi.org/10.1016/j.comcom.2004.12.045

Bengherbia, B., Ould Zmirli, M., Toubal, A., & Guessoum, A. (2017). FPGA-based wireless sensor nodes for vibration monitoring system and fault diagnosis. *Measurement: Journal of the International Measurement Confederation*, *101*, 81–92. https://doi.org/10.1016/j.measurement.2017.01.022

Bhattacharyya, S. S., Deprettere, E. F., Leupers, R., & Takala, J. (2018). *Handbook of signal processing systems*. Springer.

Bobda, C. (2008). *Introduction to reconfigurable computing: Architectures, algorithms, and applications*. *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*. https://doi.org/10.1007/978-1-4020-6100-4

Brassai, S. T., Hajdu, S., Tamas, T., & Bakó, L. (2015). Hardware implemented adaptive neuro fuzzy system. In *Carpathian Control Conference (ICCC), 2015 16th International* (pp. 58–63). IEEE.

Capability, M. (2016). 7 Series FPGAs Overview Summary of 7 Series FPGA Features Table 1 : 7 Series Families Comparison Spartan-7 FPGA Feature Summary, *180*, 1–18.

Cardona, L. A., Lorente, B., & Ferrer, C. (2014). Partial crypto-reconfiguration of nodes based on FPGA for WSN. *Proceedings - International Carnahan Conference on Security Technology*, *2014–Octob*(October), 14–17.

https://doi.org/10.1109/CCST.2014.6987046

Çavdar, T. (2016). PSO tuned ANFIS equalizer based on fuzzy C-means clustering algorithm. *AEU-International Journal of Electronics and Communications*, *70*(6), 799–807.

Chen, C., John, R., Twycross, J., & Garibaldi, J. M. (2016). An extended ANFIS architecture and its learning properties for type-1 and interval type-2 models. In *Fuzzy Systems (FUZZ-IEEE), 2016 IEEE International Conference on* (pp. 602–609). IEEE.

Chen, D., Cong, J., & Pan, P. (2006). *FPGA Design Automation: A Survey* (Vol. 3). https://doi.org/10.1561/1000000003

Chen, G., & Pham, T. T. (2000). *Introduction to fuzzy sets, fuzzy logic, and fuzzy control systems*. CRC press.

Chicago. (2015a). Air Humidity of Chicago City. Retrieved from https://data.cityofchicago.org/Parks-Recreation/Beach-Weather-Stations-Automated-Sensors-2015-Humi/4tf5-5fw5

Chicago. (2015b). Air Temperature of Chicago City. Retrieved from https://data.cityofchicago.org/Parks-Recreation/Beach-Weather-Stations-Automated- Sensors-2015-Air-/rsk3-iyyk

Chinchansure, P. S., & Kulkarni, C. V. (2014). Home Automation System based on FPGA and GSM, 3–7.

Corporation, A. (2004). Stratix GX FPGA Family Data Sheet. *Options*, (December), 1–262.

Corporation, A. (2012). 1 . Overview for the Arria II Device Family, *1*(July), 1–16.

Cyclone, T., & Partners, A. M. (2003). Features ..., (April), 1–97.

D, D. G. X. N. D. (2017). DQG : HE 7HFKQRORJLHV, *181*, 588–595. https://doi.org/10.1016/j.proeng.2017.02.438

Dalgleish, T., Williams, J. M. G. ., Golden, A.-M. J., Perkins, N., Barrett, L. F., Barnard, P. J., … Watkins, E. (2007). The Handbook of Brain Theory and Neural Networks. *Journal of Experimental Psychology: General*, *136*(1), 23–42.

Darvill, J., Tisan, A., & Cirstea, M. (2017). A novel ANFIS algorithm architecture for FPGA implementation. In *IEEE International Symposium on Industrial Electronics* (pp. 1243–1248). IEEE. https://doi.org/10.1109/ISIE.2017.8001423

Dreyfus, G. (2005). *Neural networks: methodology and applications*. Springer Science & Business Media.

Engineering, E. (2018). Application of Artificial Intelligence in Electrical Engineering, (March), 105–111.

Fausett, L. V, & Fausett, L. (1994). *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall. Retrieved from https://books.google.iq/books?id=ONylQgAACAAJ

Fularz, M., Kraft, M., Schmidt, A., & Kasi, A. (2015). Progress in Automation, Robotics and Measuring Techniques, *351*(May). https://doi.org/10.1007/978-3-319-15847-1

Fullér, R. (2013). *Introduction to neuro-fuzzy systems* (Vol. 2). Springer Science & Business Media.

Gaikwad, P. P., Gabhane, J. P., & Golait, S. S. (2015). A survey based on Smart Homes system using Internet-of-Things. In *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)* (pp. 0330–0335). IEEE. https://doi.org/10.1109/ICCPEIC.2015.7259486

Galushkin, A. I. (2007). *Neural networks theory*. Springer Science & Business Media.

Ghasemi, E., Kalhori, H., & Bagherpour, R. (2016). A new hybrid ANFIS–PSO model for prediction of peak particle velocity due to bench blasting. *Engineering with Computers*, *32*(4), 607–614.

Gomes, T., Pinto, S., Salgado, F., Tavares, A., & Cabral, J. (2017). Building IEEE 802.15.4 Accelerators for Heterogeneous Wireless Sensor Nodes. *IEEE Sensors Letters*, *1*(1), 1–4. https://doi.org/10.1109/LSENS.2017.2681625

Gómez-Castañeda, F., Tornez-Xavier, G. M., Flores-Nava, L. M., Arellano-Cárdenas, O., & Moreno-Cadenas, J. A. (2014). Photovoltaic panel emulator in FPGA technology using ANFIS approach. In *Electrical Engineering, Computing Science and Automatic Control (CCE), 2014 11th International Conference on* (pp. 1–6). IEEE.

Graupe, D. (2013). *Principles of artificial neural networks* (Vol. 7). World Scientific.

Guinn, J. R., Riedel, J. E., Bhaskaran, S., Park, R. S., Vaughan, A. T., Owen, W. M., … Martin-Mur, T. (2016). The deep-space positioning system concept: Automating complex navigation operations beyond the earth. In *AIAA SPACE 2016* (p. 5409).

Hinkelmann, H., Reinhardt, A., Varyani, S., & Glesner, M. (2008). A Reconfigurable Prototyping Platform for Smart Sensor Networks. *2008 4th Southern Conference on Programmable Logic*, *1*, 125–130. https://doi.org/10.1109/SPL.2008.4547743

Ilyas, M., & Mahgoub, I. (2004). *Handbook of sensor networks: compact wireless and wired sensing systems*. CRC press.

Imran, M., Khursheed, K., O'Nils, M., & Lawal, N. (2010). Exploration of target architecture for a wireless camera based sensor node. *Norchip*, *2010*, 1–4. https://doi.org/10.1109/NORCHIP.2010.5669490

Irwin, J. D., Wilamowski, B., & Irwin, J. D. (2011). *The Industrial Electronics Handbook*. *Eecs.Ucf.Edu*. CRC Press. https://doi.org/10.1017/CBO9781107415324.004

Jain, L. C., & Martin, N. M. (1998). *Fusion of neural networks, fuzzy systems and genetic algorithms: industrial applications* (Vol. 4). CRC press.

Jamshidi, M., & Zilouchian, A. (2001). *Intelligent control systems using soft computing methodologies*. CRC press.

Jang, J.-S. (1993). ANFIS: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(3), 665–685.

Jiesheng Wei, Ling Wang, Feng Wu, Yibo Chen, & Long Ju. (2009). Design and implementation of wireless sensor node based on open core. *2009 IEEE Youth Conference on Information, Computing and Telecommunication*, 102–105. https://doi.org/10.1109/YCICT.2009.5382416

John, M., & Smith, S. (n.d.). *Application-Specific Integrated Circuits* (First Edit). Addison Wesley Longman 1997.

June, D. S., & Ram, B. (2013). Spartan-3 FPGA Family Module 1 : Module 4 : Pinout Descriptions Module 2 : Functional Description Spartan-3 FPGA Family :, 1–272.

Kacprzyk, J., & Pedrycz, W. (2015). *Springer handbook of computational intelligence*. Springer.

Karakuzu, C., Karakaya, F., & Çavuşlu, M. A. (2016). FPGA implementation of neuro-fuzzy system with improved PSO learning. *Neural Networks*, *79*, 128–140.

Karl, H., & Willig, A. (2007). *Protocols and architectures for wireless sensor networks*. John Wiley & Sons.

Kaur, G., Dhar, J., & Guha, R. K. (2016). Minimal variability OWA operator combining ANFIS and fuzzy c-means for forecasting BSE index. *Mathematics and Computers in Simulation*, *122*, 69–80.

Kaynak, O., Zadeh, L. A., Turksen, B., & Rudas, I. J. (1996). *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications* (Vol. 162). Springer Science & Business Media.

Kaynak, O., Zadeh, L. A., Türksen, B., & Rudas, I. J. (2012). *Computational intelligence: Soft computing and fuzzy-neuro integration with applications* (Vol. 162). Springer Science & Business Media.

Kelly, B. M., Rumberg, B., Graham, D. W., & Kulathumani, V. (2013). Reconfigurable analog signal processing for wireless sensor networks. *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 221–224. https://doi.org/10.1109/MWSCAS.2013.6674625

Kerhet, A., Magno, M., Leonardi, F., Boni, A., & Benini, L. (2007). A low-power wireless video sensor node for distributed object detection. *Journal of Real-Time Image Processing*, *2*(4), 331–342. https://doi.org/10.1007/s11554-007-0048-7

Khedkar, A. A., & Khade, R. H. (2017). High speed FPGA-based data acquisition system. *Microprocessors and Microsystems*, *49*, 87–94.

Konar, A. (1999). *Artificial intelligence and soft computing: behavioral and cognitive modeling of the human brain*. CRC press.

Krasteva, Y. E., Portilla, J., Carnicer, J. M., De La Torre, E., & Riesgo, T. (2008). Remote HW-SW reconfigurable wireless sensor nodes. *IECON Proceedings (Industrial Electronics Conference)*, 2483–2488. https://doi.org/10.1109/IECON.2008.4758346

Kumar, S., Shegokar, S., Chouhan, D., & Iqbal, A. (2018). Different Technology Comparison for Home Automation System, *1*(9), 2–6. Retrieved from http://irejournals.com/FormDetails.jsp?id=1700231

Łapa, K., Zalasiński, M., & Cpałka, K. (2013). A new method for designing and complexity reduction of neuro-fuzzy systems for nonlinear modelling. In *International Conference on Artificial Intelligence and Soft Computing* (pp. 329–344). Springer.

Lee, E. A., & Seshia, S. A. (2011). *Embedded Systems*.

Li, J., Xu, T., & Zhang, K. (2017). Real-time feature-based video stabilization on FPGA. *IEEE Transactions on Circuits and Systems for Video Technology*, *27*(4), 907–919.

Liao, J., Singh, B. K., Khalid, M. A. S., & Tepe, K. E. (2013). Open Access FPGA based wireless sensor node with customizable event-driven architecture, 1–11.

Liu, P., & Li, H.-X. (2004). *Fuzzy neural network theory and application* (Vol. 59). World Scientific.

Liu, W., Bienstman, L., Jooris, B., Yaron, O., & Moerman, I. (2012). FPGA-Based Wireless Link Emulator, 48–63.

Lopez, B., Valverde, J., De La Torre, E., & Riesgo, T. (2014). Power-aware multi-objective evolvable hardware system on an FPGA. *Proceedings of the 2014 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2014*, 61–68. https://doi.org/10.1109/AHS.2014.6880159

Macmahon, S., & Cherukupaly, U. (2014). LightWeight IP Application Examples, *1026*, 1–31.

Manual, T. R. (2015). ARM ® Cortex ® -A72 MPCore Processor, 1–577.

Manual, T. R. (2018). Zynq-7000 SoC, *585*.

Marufuzzaman, M., Bin, M., Reaz, I., & Islam, M. T. (2014). FPGA based Distributed Task Organizing Agents in Smart Home. In *Proc. of the 2nd Int. Conf. on Research in Science, Engineering and Technology, Dubai* (pp. 2–5). https://doi.org/10.15242/IIE.E0314504

Mas, T. T. Ă., & Brassai, S. T. (2015). Hardware Implementation of a Neuro-Fuzzy Controller Using High Level Synthesis Tool. *MACRo 2015*, *1*(1), 183–191.

Maxfield, C. (2004). *The Design Warrior ' s Guide to FPGAs*. *System*.

Meena, S., & Krishna Prakash, N. (2014). Simulation of Dynamically Reconfigurable Wireless Sensor Node. *2014 International Conference on Electronics and Communication System (LCECS -2014).*

Mehrotra, K., Mohan, C. K., & Ranka, S. (1997). *Elements of artificial neural networks*. MIT press.

Muralidhar, P., & Rao, C. B. R. (2008). Reconfigurable Wireless sensor network node based on NIOS core. *Proceedings of the 4th International Conference on Wireless Communication and Sensor Networks, WCSN 2008*, 67–72.

https://doi.org/10.1109/WCSN.2008.4772684

Nadu, T., & Nadu, T. (2017). Fpga Implementation Of Neuro Fuzzy System With Adaptive Particle Swarm Optimization Learning For Image Edge Detection, 181–185.

Nikam, S. R., Nikumbh, P. J., & Kulkarni, S. P. (2012). Fuzzy logic and neuro-fuzzy modeling. In *MPGI National Multi Conference* (pp. 7–8).

Panigrahi, A., Qureshi, A., Saxena, P., & Reddy, E. R. (2016). Home Automation Using FPGA Controller, (5), 1064–1066.

Philipp, F. (2014). Runtime Hardware Reconfiguration in Wireless Sensor Networks for Condition Monitoring. Universitäts-und Landesbibliothek Darmstadt.

Philipp, F., & Glesner, M. (2011). Mechanisms and Architecture for the Dynamic Reconfiguration of an Advanced Wireless Sensor Node. *2011 21st International Conference on Field Programmable Logic and Applications*, 396–398. https://doi.org/10.1109/FPL.2011.78

Poudel, B., Giri, N. K., & Munir, A. (2017). Design and comparative evaluation of GPGPU-and FPGA-based MPSoC ECU architectures for secure, dependable, and real-time automotive CPS. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)* (pp. 29–36). IEEE.

PowWow: Power Optimized Hardware and Software FrameWork for Wireless Motes. (2018), (2010), 2018.

Prabhu, V., Jena, J., & Rode, S. (2018). Home automation using artificial intelligence, 780–784. https://doi.org/10.23883/IJRTER.2018.4206.PD8PQ

Prakash, N. K., & Surjith, B. (2017). FPGA Based Remote Monitoring System in Smart Grids. *Indian Journal of Science and Technology*, *10*(5). https://doi.org/10.17485/ijst/2017/v10i5/108829

Rahimi, M., Estrin, D., Baer, R., Uyeno, H., & Warrior, J. (2004). Cyclops, image sensing and interpretation in wireless networks. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems - SenSys '04*, 311. https://doi.org/10.1145/1031495.1031554

Reaz, M. B. I. (2013). Artificial Intelligence Techniques For Advanced Smart Home Implementation. *Acta Technica Corvininesis-Bulletin of Engineering*, *6*(2).

Reaz, M. B. I., Assim, A., Choong, F., Hussain, M. S., & Mohd-Yasin, F. (2006). Multiagent System for Home Automation. In *WSEAS International Conference on Signal Processing (SIP '06)* (pp. 135–139). Citeseer. Retrieved from https://dl.acm.org/citation.cfm?id=1983968

Reference, A. X. I., Axi, V., & Guide, R. (2014). Vivado Design, *1037*, 1–143.

Rini, D. P., Shamsuddin, S. M., & Yuhaniz, S. S. (2016). Particle swarm optimization for ANFIS interpretability and accuracy. *Soft Computing*, *20*(1), 251–262.

Rusu, O.-V., & Duka, A.-V. (2017). Monitoring and Control Platform for Homes Based on FPGA, SoC and Web Technologies. *Procedia Engineering*, *181*, 588–

595.

Sahlbach, H., Thiele, D., & Ernst, R. (2017). A system-level FPGA design methodology for video applications with weakly-programmable hardware components. *Journal of Real-Time Image Processing*, *13*(2), 291–309.

Sang-hyun, L., Lee, J., & Kyung-il, M. (2013). Smart Home Security System Using Multiple ANFIS, *7*(3), 121–132.

Sarpotdar, M., Mathew, J., Safonova, M., & Murthy, J. (2016). A generic FPGA-based detector readout and real-time image processing board. In *High Energy, Optical, and Infrared Detectors for Astronomy VII* (Vol. 9915, p. 99152K). International Society for Optics and Photonics.

Schrader, T., Bredemeyer, J., Mihalachi, M., Rohde, J., & Kleine-Ostmann, T. (2016). Concept and design of a UAS-based platform for measurements of RF signal-in-space. *Advances in Radio Science*, *14*(A.), 1–9.

Sharma, S., Boddu, J., Charan, G. S., Sharma, S., Sivanantham, S., & Sivasankaran, K. (2015). Home automation through FPGA controller. In *Green Engineering and Technologies (IC-GET), 2015 Online International Conference on* (pp. 1–4). IEEE.

Siddique, N., & Adeli, H. (2013). *Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing*. John Wiley & Sons.

Sivanandam, S. N., Sumathi, S., & Deepa, S. N. (2007). *Introduction to fuzzy logic using MATLAB* (Vol. 1). Springer.

Sohraby, K., Minoli, D., & Znati, T. (2007). *Wireless sensor networks: technology, protocols, and applications*. John Wiley & Sons.

Sumathi, S., & Paneerselvam, S. (2010). *Computational intelligence paradigms: theory & applications using MATLAB*. CRC Press.

Suresh, G. (2018). Effective Replacement of FPGA for Microcontrollers in Home Automation, *13*(5), 2710–2713.

Suresh, G., & Mastani, S. A. (2018). Effective Replacement of FPGA for Microcontrollers in Home Automation. *International Journal of Applied Engineering Research*, *13*(5), 2710–2713. Retrieved from https://www.ripublication.com/ijaer18/ijaerv13n5_82.pdf

Synthesis, H. (2016). Vivado Design Suite User Guide, *902*.

Systems, M., & Smart, F. O. R. (n.d.). Artificial Intelligence Techniques For Advanced.

Szilvási, S., Babják, B., Völgyesi, P., & Lédeczi, Á. (2013). Marmote SDR: Experimental Platform for Low-PowerWirelessProtocol Stack Research. *Journal of Sensor and Actuator Networks*, *2*(3), 631–652. https://doi.org/10.3390/jsan2030631

Tămas, T., & Brassai, S. T. (2015). Hardware implementation of a neuro-fuzzy controller using high level synthesis tool. *MACRo 2015*, *1*(1), 183–191.

Tettamanzi, A., & Tomassini, M. (2013). *Soft computing: integrating evolutionary, neural, and fuzzy systems*. Springer Science & Business Media.

Velez, G., & Otaegui, O. (2016). Embedding vision-based advanced driver assistance systems: a survey. *IET Intelligent Transport Systems*, *11*(3), 103–112.

Vieira, J., Dias, F. M., & Mota, A. (2004). Neuro-fuzzy systems: a survey. In *5th WSEAS NNA international conference on neural networks and applications, Udine, Italia*.

Waghmare, P., Chandgude, M., Chaure, P., & Chaudhari, A. (2017). Survey on : Home Automation Systems, 7–10.

Waghmare, P., Chaure, P., Chandgude, M., & Chaudhari, A. (2017). Survey on: Home automation systems. In *2017 International Conference on Trends in Electronics and Informatics (ICEI)* (pp. 7–10). https://doi.org/10.1109/ICOEI.2017.8300864

Wilder, J. L., Uzelac, V., Milenković, A., & Jovanov, E. (2008). Runtime hardware reconfiguration in wireless sensor networks. *Proceedings of the Annual Southeastern Symposium on System Theory*, (September), 154–158. https://doi.org/10.1109/SSST.2008.4480210

Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity, 1–45. https://doi.org/10.1080/02673039508720837

Woods, R., McAllister, J., Lightbody, G., & Yi, Y. (2017). *FPGA-based implementation of signal processing systems*. Wiley Online Library.

Xilinx. (2011). Spartan-6 Family Overview Summary of Spartan-6 FPGA Features. *Product Specification*, *160*(DS160 v2.0), 1–11.

Xilinx. (2016). lwIP 1.4.0 Library, *v1.7*(UG650), 15.

Xilinx. (2018). Zynq-7000 SoC Data Sheet : Overview, *190*, 1–25.

Xilinx Inc. (2013). Virtex-7 T and XT FPGAs Data Sheet : DC and AC Switching Characteristics, *183*, 62.

Xilinx Inc. SDK. (2016). Vivado Design Suite User Guide, *898*, 1–156.

Yang, M., Liu, B., Gong, J., Liu, H., Hu, H., Dong, Y., … Miao, Z. (2016). Architecture design for reliable and reconfigurable FPGA-based GNC computer for deep space exploration. *Science China Technological Sciences*, *59*(2), 289–300.

Yonezawa, Y., Nakao, H., Nakashima, Y., Vithanage, A., Kanehira, T., & Ueno, Y. (2017). Model-based development of high-current-density point-of-load converter of high performance FPGA for telecommunication application. In *2017 IEEE International Telecommunications Energy Conference (INTELEC)* (pp. 351–356). IEEE.

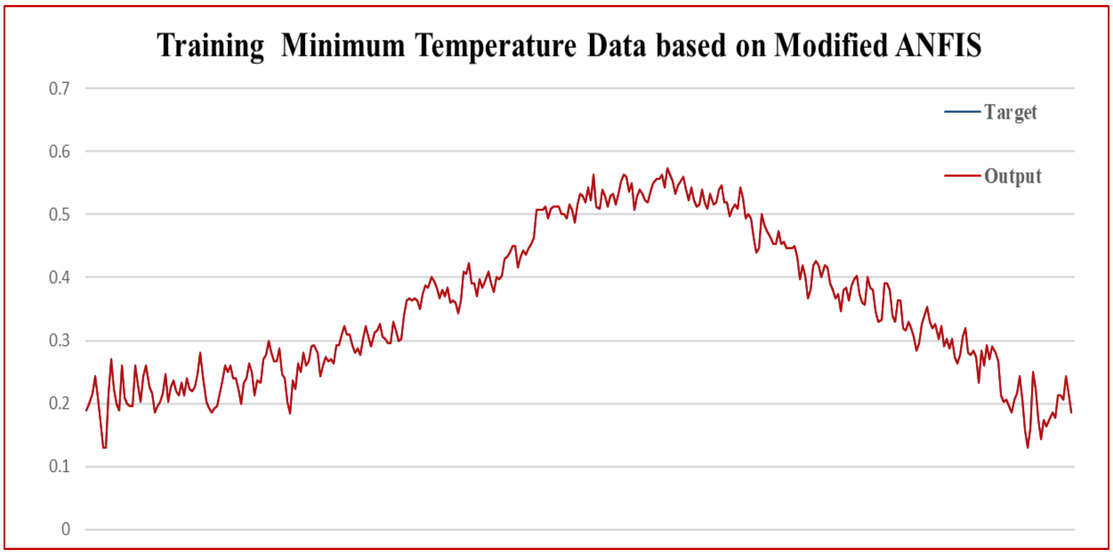
Yuan, S., Qiu, L., Gao, S., Tong, Y., & Yang, W. (2012). Providing self-healing ability for wireless sensor node by using reconfigurable hardware. *Sensors (Switzerland)*, *12*(11), 14570–14591. https://doi.org/10.3390/s121114570

Yugashini, I., Vidhyasri, S., & Devi, K. G. (2013). Design And Implementation Of Automated Door Accessing System With Face Recognition, (12), 10–13.

Zhai, X., Ali, A. A. S., Amira, A., & Bensaali, F. (2016). MLP Neural Network Based Gas Classification System on Zynq SoC. *IEEE Access*, *4*, 8138–8146. https://doi.org/10.1109/ACCESS.2016.2619181

Appendix A

A.1. Training the Modified Algorithm and ANFIS Toolbox based on Minimum Temperature and Humidity Data Sets



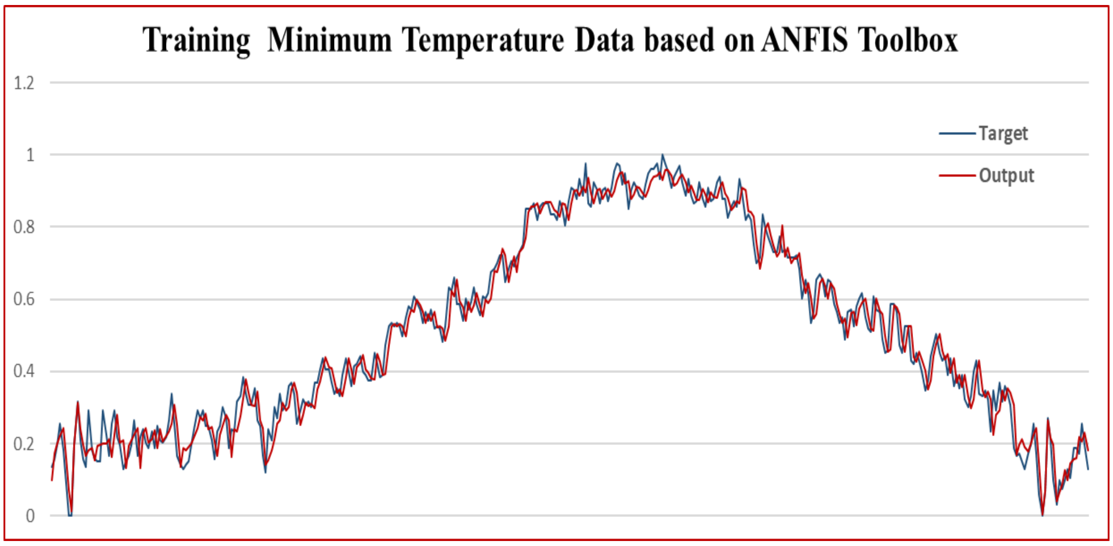
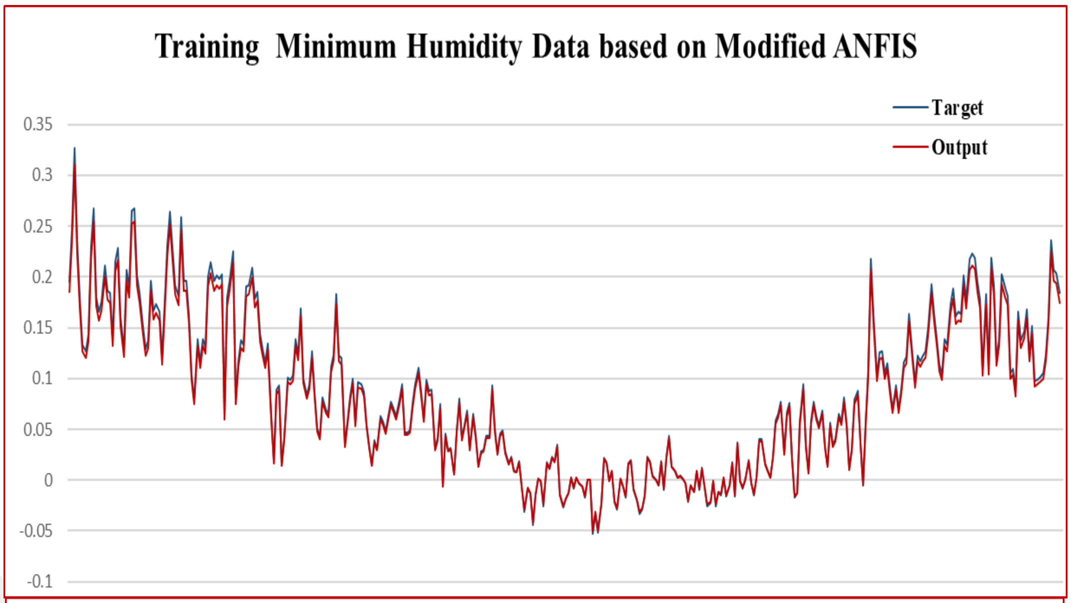


Figure A.01. Training Minimum Temperature Data



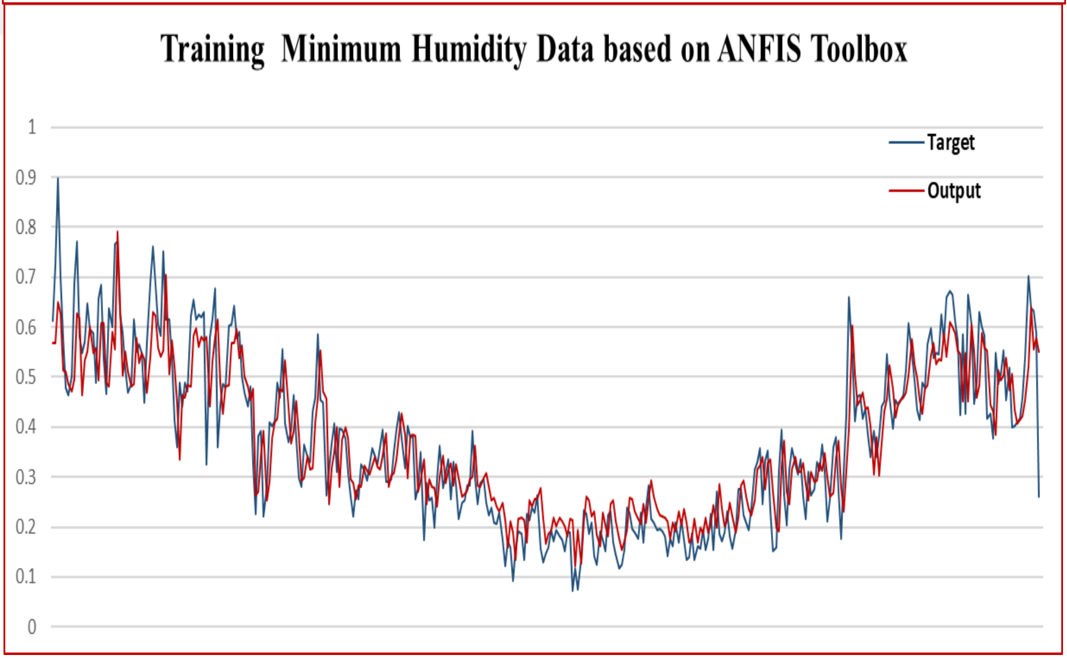
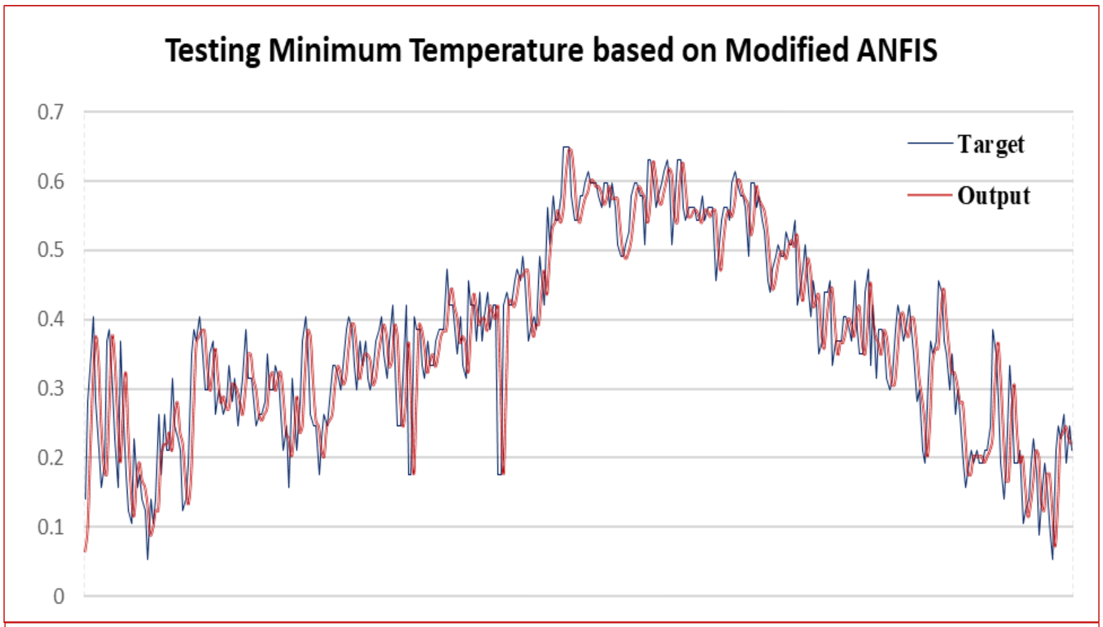


Figure A.3. Training Minimum Humidity Data set

A.2. Testing the Modified Algorithm and ANFIS Toolbox based on Minimum Temperature and Humidity Data Sets



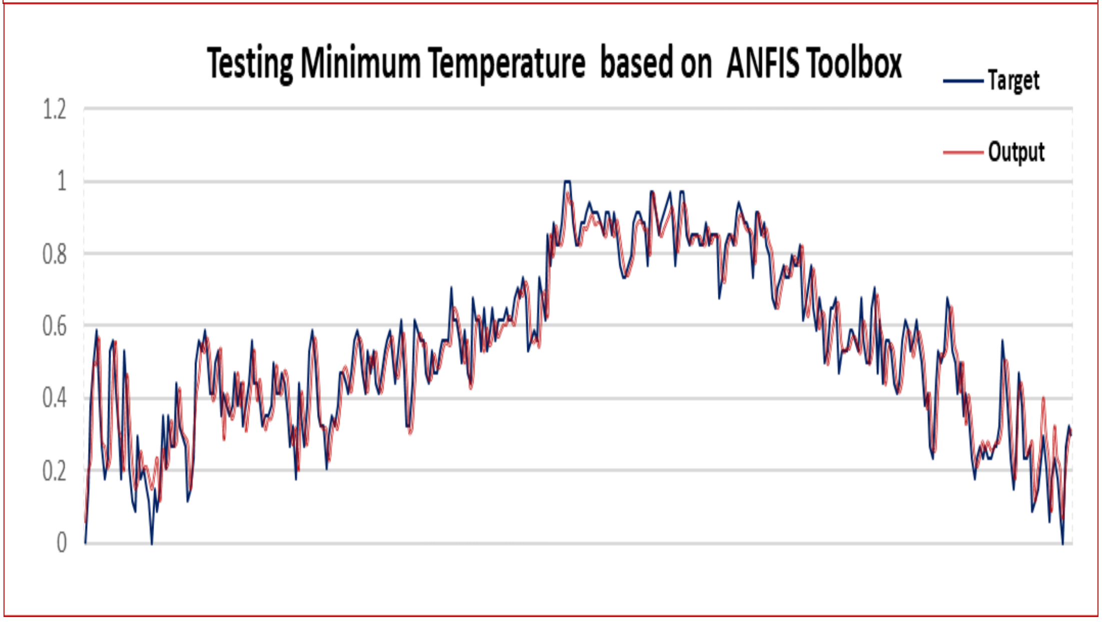


Figure A.5. Testing Minimum Temperature Data set

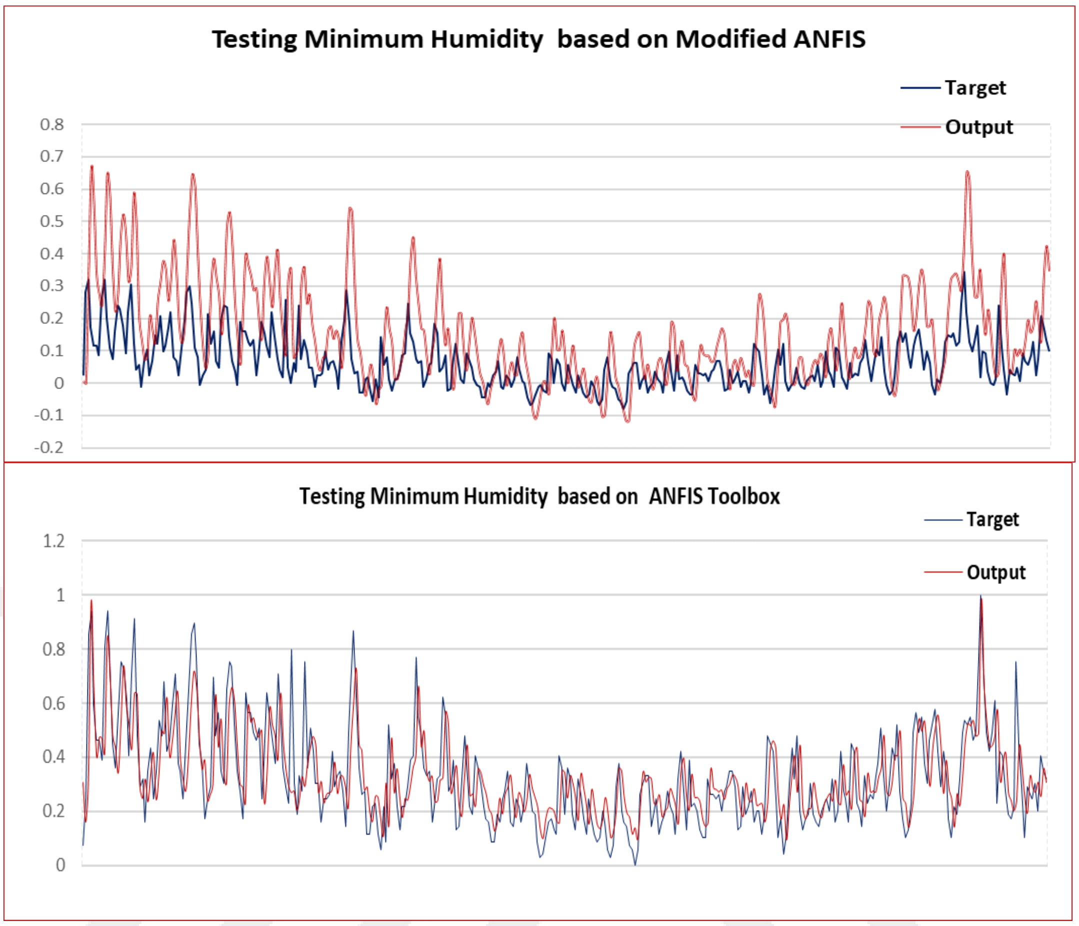
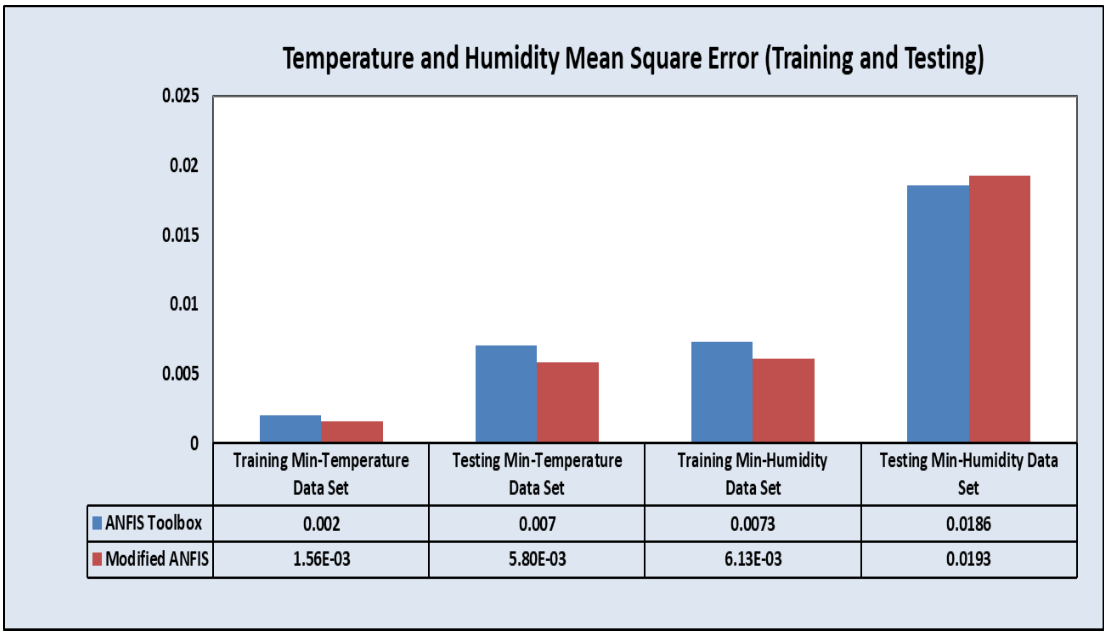


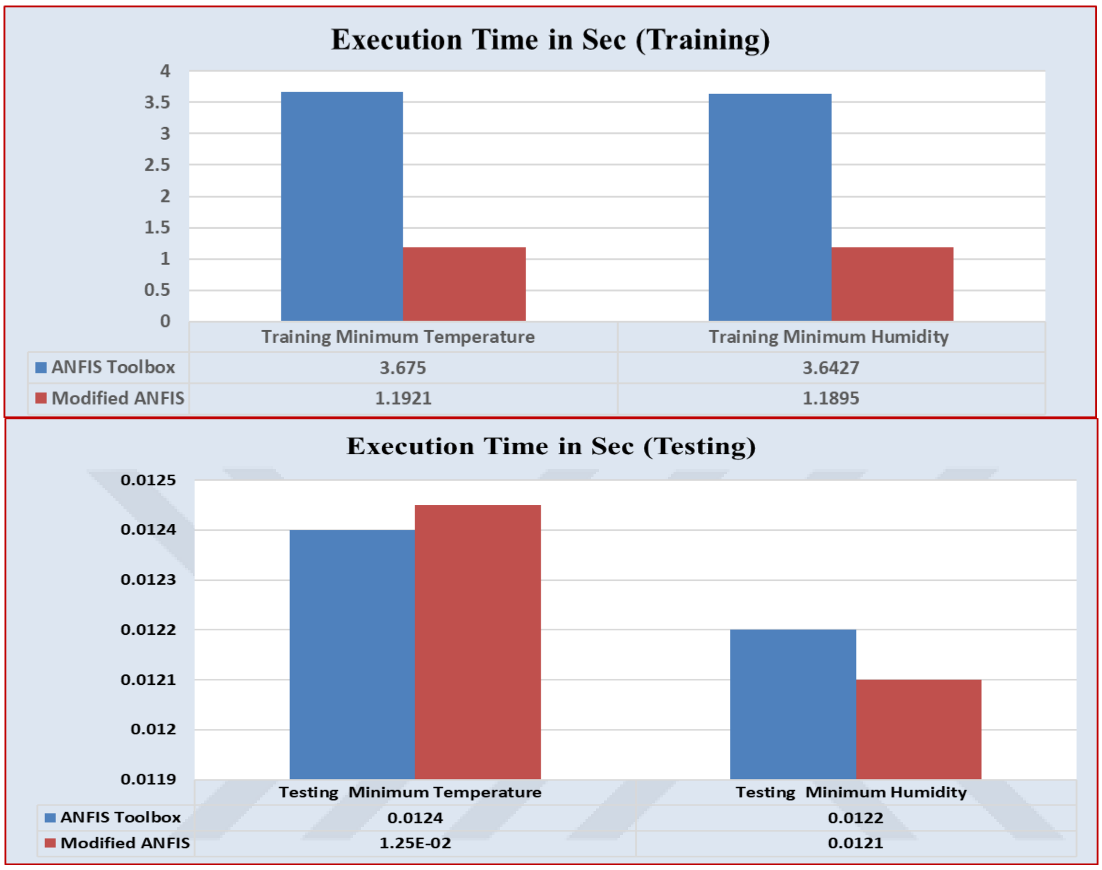
Figure A.7. Testing Minimum Humidity Data Set



	Training Min-Temperature Data Set	Testing Min-Temperature Data Set	Training Min-Humidity Data Set	Testing Min-Humidity Data Set
🟦 ANFIS Toolbox	0.002	0.007	0.0073	0.0186
🟥 Modified ANFIS	1.56E-03	5.80E-03	6.13E-03	0.0193

Figure A.9. MSE of Training and Testing Minimum Temperature and Humidity Sets

A.3. Training and Testing Execution Time



	Training Minimum Temperature	Training Minimum Humidity
🟦 ANFIS Toolbox	3.675	3.6427
🟥 Modified ANFIS	1.1921	1.1895

	Testing Minimum Temperature	Testing Minimum Humidity
🟦 ANFIS Toolbox	0.0124	0.0122
🟥 Modified ANFIS	1.25E-02	0.0121

Figure A.11. Training and Testing Execution Time