People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering
Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of the Requirements of the Degree of

'MASTER'

In: **Computer Engineering**

Title:

FPGA-based Artificial Neural Network driving a Stepper Motor

Presented By:

- **OUDJEHANI Celina**
- **BENNOUR Khadidja**

Supervisor:

Dr. A. Benzekri

Registration Number: 2021/2022

ACKNOWLEDGEMENT

We would like to express our profound gratitude to our supervisor Dr.A.BENZEKRI for his guidance and support throughout the making of this project.

A debt of gratitude is owed to our beloved parents, without whom none of this would have been possible. We also thank our siblings, family and friends for their aid and encouragements.

ABSTRACT

This report describes the design and implementation of an FPGA-based Artificial Neural Network (ANN) for character recognition. The ANN algorithm is fully developed using VHDL in the structural modelling style. It comprises of 16 nodes in the input layer, 32 in the hidden layer and 16 in the output layer. The processing of data is done in the IEEE single precision floating-point format. In order to train the ANN, a dataset of 4×4 matrices stored in a VHDL file is used to represent the 16 letters to be recognized: A, C, D, F, H, I, J, L, N, O, P, T, U, X, Y, Z that are selected based on the feasibility of their representation in such dimensions. The weights are randomly initialized with a 16-bit Galois LFSR that has a maximum period of 65535, which are stored in an on-board SRAM unit of 2MB storage capacity. The DE2-115 board hardware platform is utilized to synthesize the overall system with the Quartus II software version 13.0. The built-in LCD display serves as an interface for the user to input the desired pattern on the two 4×4 grids, as well as to show the output class of the recognition process. We added a stepper motor circuit to test the working of the ANN with "ON" and "OF" patterns.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Aclr	Asynchronous Clear
ADAS	Advanced Driver Assistance Systems
ADC	Analog to Digital Converter
AGI	Artificial General Intelligence
AI	Artificial Intelligence
ALU	Arithmetic-Logic Unit
ANN	Artificial Neural Networks
ASI	Artificial Superintelligence
ASIC	Application Specific Integrated Circuits
ASSP	Application Specific Standard Products
BRAM	Block Random Access Memory
CLB	Configurable Logic Blocks
clk_en	Clock Enable
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CT scans	Computerized Tomography Scans
DAC	Digital to Analog Converter
Dl	Deep Learning
DRBG	Deterministic Random Bit Generator
DSP	Digital Signal Processing
FF	Flip-Flops
FFT	Fast Fourier Transforms
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPIO	General Purpose Input/Output
GPP	General Purpose Processors
GPU	Graphics Processing Unit

HDL	Hardware Description Language
HLS	High-Level Synthesis
I P	Intellectual Property
I/O	Input/Output
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
LCD	Liquid Crystal Display
LE	Logic Element
LED	Light Emitting Diode
LFSR	Linear Feedback Shift Register
LUT	Look-up Table
ML	Machine Learning
MSE	Mean Squared Error
MUX	Multiplexer
NaN	Not-a-Number
NLP	Natural Language Processing
OCR	Optical Character Readers
PCB	Printed Circuit Boards
PET	Positron Emission Tomography
PLD	Programmable Logic Device
PR	Pattern Recognizer
PRNG	Pseudo-Random Numbers Generator
PROMs	Programmable Read-Only Memories
RAM	Random Access Memory
RTL	Register-Transfer Level
SDR	Software-Defined Radio
SoC	System-On-Chip
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory

SVM	Support Vector Machines
TPU	Tensor Processing Unit
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Chapter 01

Introduction

This chapter provides an overview of the project, its motivation and objectives. It also describes the system's structure and the report's organization.

1. OVERVIEW

In the age of artificial intelligence, machine learning takes the spotlight as a major topic of interest. Nobody could have foreseen the dimension of impact computer vision and predictive analytics would have on our day to day life. Whether it be self driving vehicles, speech, face recognition or AIs accomplishing everyday mundane tasks, what used to be science fiction is today, considered to be within reach.

Machine learning is probably the most well-known and widely used AI technology. It uses large amounts of historical data to create machines that learn and enhance themselves through experience, much like humans do via learning and observation. And as the constant need for data analysis increases with the sheer quantity and pace at which individuals and enterprises generate data, machine learning is largely solicited.

Data has become the equivalent of gold, in the 21st century; it has been propelled to the status of valuable commodity. Businesses are actively and constantly looking for efficient and innovative ways to assess their data and reveal insights that could aid in the improvement of their business processes and decision making. This pushes them to seek Artificial Intelligence (AI) systems and technologies which enable them to automate business processes and make better and more informed decisions.

AI and DL applications require a large amount of data. This is why domain experts and industry practitioners associate them with the use of significant amounts of computing resources such as Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) processors. This is largely due to the fact that non-trivial deep learning applications include complex linear algebraic computations like matrix and vector operations. GPUs and TPUs are extremely efficient and fast at performing such operations, making them ideal for running deep learning algorithms [1].

Modern cloud computing is the primary infrastructure employed in running machine learning applications. Cloud computing can also be thought of as utility computing or on-demand computing, and it provides access to large amounts of computing resources, thus its use in machine learning.

However, executing AI in the cloud is not always the best option, particularly for applications that require low latency and must be run close to end users. The benefits of bringing AI closer to users are felt on a daily basis by millions of consumers who use deep learning applications in their smart phones, such as Siri, OK Google, and Apple FaceID.

This is where embedded machine learning comes into focus, as it liberates the previously untapped potential of enterprise data. As with smart phones, machine learning models can be executed on a wide range of embedded devices, from networked and mobile embedded systems to small scale microcontrollers. Embedded ML works on the general principle that ML models, such as neural networks, are trained on computing clusters or in the cloud, while inference operations and model execution occur on embedded devices. Once trained, deep learning models' matrix operations can be effectively executed on CPU (Central Processing Unit) constrained devices or even tiny (e.g., 16 or 32 bit) microcontrollers.

Embedded machine learning unleashes the power of data processing within the hundreds of billions of readily available microprocessors and embedded controllers found in a wide range of settings such as industrial plants, smart buildings, and residential environments. Throughout this way, it also facilitates the processing of data generated by embedded devices (for example, Internet of Things devices), the majority of which are currently underutilized.

The execution of machine learning models on embedded devices has several advantages over cloud-based AI:

- Low Latency: When low-latency operations must be performed close to the field, embedded machine learning is far more efficient than cloud AI. This is because there is no need to send large amounts of data to the cloud, which can cause significant network latency. As a result, embedded machine learning is an excellent choice for supporting real-time use cases.
- Network Bandwidth Efficiency: Running machine learning models on embedded devices allows for the extraction of features and insights at the data source. This eliminates the need to transfer raw data to edge or cloud servers, saving bandwidth and network resources.
- Improved Environmental Performance: Cloud AI results in CO2 emissions and has very poor environmental performance. Machine learning on embedded devices, on the other hand, has a far reduced carbon footprint, resulting in much higher sustainability [1].
- Strong Privacy: Embedded machine learning eliminates the requirement for data to be transferred and stored on cloud servers. This decreases the likelihood of data breaches and privacy leaks, which is especially critical for apps that process sensitive data such as citizens' personal information, intellectual property (IP) data, and company secrets.

Overall, an ecosystem of hardware and software assets exists to support the creation, deployment, and operation of embedded machine learning applications. As more developers and integrators ride the wave of embedded machine learning applications, this ecosystem is expanding. There is already a diverse variety of embedded devices capable of running machine learning and deep learning applications. Many gadgets are low-cost and can be used in a variety of IoT applications and the FPGA is one of them [1].

1.1 Machine Learning and Deep Learning on FPGA

Historically, when evaluating hardware platforms for acceleration, the trade-off between flexibility and performance must be considered. On one end of the spectrum, general purpose processors (GPP) offer a high level of flexibility and ease of use, but perform inefficiently. These platforms are more widely available, less expensive to produce, and suitable for a wide range of uses and reuses. On the other end of the spectrum, application specific integrated circuits (ASIC) provide high performance at the expense of being more rigid and difficult to construct. These circuits are designed for a specific application and are costly and time consuming to manufacture. FPGAs are a middle ground between these two extremes.

They are a type of programmable logic device (PLD) and, in the most basic sense, a reconfigurable integrated circuit. As such, they combine the benefits of integrated circuit performance with the reconfigurable flexibility of GPPs. FPGAs can implement sequential logic using flip-flops (FF) and combinational logic using look-up tables at a low level (LUT) [2].

Hardened components for commonly used functions such as full processor cores, communication cores, arithmetic cores, and block RAM are also included in modern FPGAs (BRAM). Furthermore, current FPGA trends point to a system-on-chip (SoC) design approach, in which ARM coprocessors and FPGAs are frequently found on the same fabric. AMD Xilinx and Intel FPGA currently dominate the FPGA market, accounting for a combined 85 percent market share [3]. Furthermore, for fixed function logic, FPGAs are rapidly replacing ASICs and application specific standard products (ASSP).

FPGAs provide an obvious potential for acceleration above and beyond what is possible on traditional GPPs for deep learning. The traditional Von Neumann architecture is used for software-level execution on GPPs, which stores instructions and data in external memory to be fetched when needed. This is the motivation behind caches, which eliminate many of the costly external memory operations [3]. The processor and memory are the bottlenecks in this architecture.

GPP performance is severely hampered by poor communication. Particularly for memory-bound techniques that are frequently required in deep learning. By contrast, the programmable FPGA logic cells can be used to implement the data and control path found in common logic functions that do not use Von Neumann architecture. They can also take advantage of distributed on-chip memory and high levels of pipeline parallelism, which fits well with the feed-forward nature of deep learning methods.

Modern FPGAs also support partial dynamic reconfiguration, which allows for the reprogramming of a portion of the FPGA while another portion of the FPGA is in use. This has implications for large deep learning models, as individual layers on the FPGA could be reconfigured without disrupting ongoing computation in other layers. This would allow for models that are too large to fit on a single FPGA while also reducing expensive global memory reads by storing intermediate results in local memory.

Above all, when compared to GPUs, FPGAs provide a unique perspective on what it means to accelerate designs on hardware. A software execution model is followed with GPUs and other fixed architectures, and it is structured around executing tasks in parallel on independent compute units.

As a result, the goal of developing deep learning techniques for GPUs is to adapt algorithms to follow this model, which ensures parallel computation and data interdependence. In contrast, FPGA architecture is application-specific. There is less emphasis on adapting algorithms for a fixed computational structure when developing deep learning techniques for FPGAs, allowing more freedom to explore algorithm level optimization Techniques that necessitate numerous complex low-level hardware control operations which are difficult to implement in high-level software Languages, are particularly appealing for FPGA implementations.

Both AMD Xilinx former Xilinx and Intel FPGA former Altera have advocated for the use of high-level design tools that abstract away many of the difficulties associated with low-level hardware programming. These tools are known as high-level synthesis (HLS) tools because they convert high-level designs into low-level register-transfer level (RTL) or HDL code.

According to AMD Xilinx research, FPGAs can produce roughly the same or more compute power than comparable GPUs. FPGAs also have more on-chip memory, which results in greater compute capability. This memory alleviates bottlenecks caused by external memory access while also lowering the cost and power required for high memory bandwidth solutions.

FPGAs can support a wide range of data types in computations, including FTP32, INT8, binary, and custom types. FPGAs can be modified as needed, whereas GPUs require vendors to adapt architectures to ensure compatibility. This may imply putting projects on hold while vendors make changes.

As for the safety aspect, GPUs were created for high-performance computing and graphics workloads. Concerns about safety were unimportant. However, GPUs have been used in applications where functional safety is a concern, such as ADAS. GPUs must be designed to meet safety requirements in these cases, which can be time-consuming for vendors.

In contrast, because FPGAs are programmable, you can design them to meet whatever safety requirements you have. These circuits have been successfully used in automation, avionics, and defense without the need for custom manufacturing.

Ultimately, FPGAs constitute a better pick for the implementation of both machine learning and deep learning applications. Amongst the several existing machine learning application, pattern recognition is our field of interest.

Pattern recognition is a data analysis technique that employs machine learning algorithms to detect patterns and regularities in data. This data can range from text and images to sounds and other definable characteristics. Pattern recognition systems can quickly and accurately recognize familiar patterns. They can also recognize and classify new objects, recognize shapes and objects from various angles, and identify patterns and objects that are partially obscured.

Remote sensing in artificial systems entails the classification of spectral data for ecosystem and land management, optical character readers (OCRs) must comprehend written text, and biometrics seeks human identity based on how people look—face, iris, and retina—or act—gait, fingerprints, and/or hand geometry. Furthermore, for safe navigation and efficient manipulation, robots must recognize obstacles such as the layout and identities of surrounding objects [4].

Previous works related to machine learning on hardware include the execution of neural network training on GPU and CPU, the use of vivado hls, python and Tenserflow and various microprocessors to achieve the desired results. The newest topics of interest are FPGA-based Artificial Neural Networks.

2. Motivation

We aim through this project to lay the groundwork for the advancement of hardware implemented machine learning applications, and more specifically pattern recognition algorithms .The FPGA has been chosen as the main tool for embedding the machine learning application, due to the various advantages it offers, from compute power to safety, as well as the worldwide interest in embedded ML's constant rise.

3. Project Objectives

The purpose of this project is to implement an artificial neural network entirely using the VHDL language, trained and deployed on the DE2-FPGA, whose main purpose is to recognize a set of patterns, more specifically some characters of the alphabet. A stepper motor will then be controlled based on specific patterns presented to the ANN as input from the user in the two 4×4 LCD grids.

-The switches of the DE2-115 FPGA, serve as a user interaction interface to input the desired patterns on the LCD.

-KEY [3] is utilized to enable the user to trigger the recognition process and KEY [0] is provided as a mean for resetting the whole system.

- The built-in LCD display provides the user for a visual interaction with the pattern recognition system.

-The PR block manages the training and running phases of the ANN.

-The onboard SRAM chip is used to store the ANN weights efficiently.

- The Linear feedback shift register allows for initializing the weights with pseudo-random numbers.

- The ANN processes the classes of the dataset sequentially for the learning stage and the two 4 × 4 LCD grids user inputs in the running stage through the employment of a floating-point ALU.

-The floating-point ALU performs the necessary arithmetic operations to achieve the recognition.

- A 28BYJ-48 stepper motor circuit is designed as a demonstration example for the accuracy of the pattern recognition system.

4. Structure of the FPGA-based pattern recognizer

The main hardware equipment employed in this project prototype is the DE2-115 FPGA board that contains the on-chip synthesized system shown on Figure 1.1, which comprises of seven interlinked elements that operate to control the off-chip hardware consisting of a stepper motor circuit.

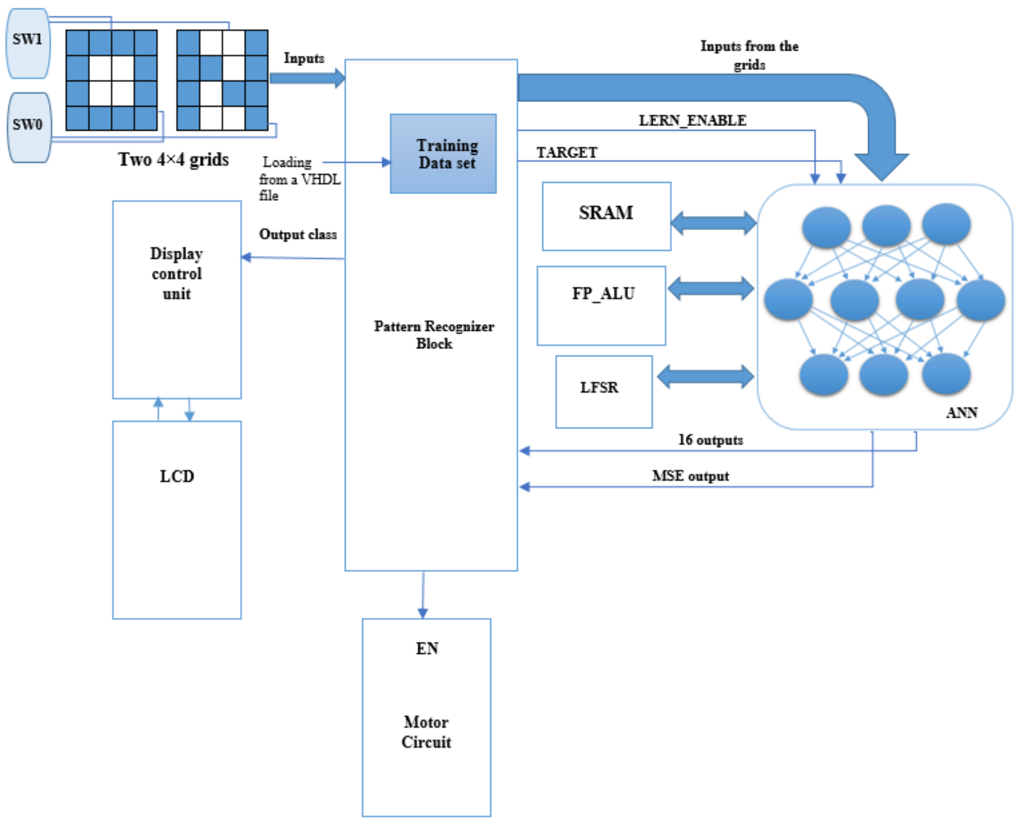


Figure 1-1: An overall design schematic of the pattern recognizer system.

5. Organization of the report

This report is partitioned into three chapters. Chapter 2 delves into the theoretical background of the project, it gives a detailed image of the science behind the main field that the work is built upon, and it looks through the totality of the components used, their mode of operation and general description. The majority of the report is contained in chapter 3. It describes the software design for the project; it includes flowcharts that describe the algorithms used in the system’s configuration. It also relates in detail the design and implementation of the project's hardware, as well as the interface between the various system-components. The conclusion summarizes the work presented in this report, discusses the findings, and makes recommendations for future research. The report ends with a list of references for further reading on the subject.

Chapter 02

Theoretical background

This chapter explores the theoretical background of the project; it provides a detailed image of the science behind the main area on which the work is based. It also examines the totality of the components used, as well as their mode of operation and general description.

1. Artificial Intelligence

Artificial Intelligence (AI) is a field of study in computer science. It entails creating computer programs to perform tasks that would otherwise require human intelligence.

AI algorithms are used in a variety of applications in the modern world, as they can tackle multiple situations involving problem solving, and are used to achieve machine autonomy. In general, AI systems operate by in taking large amounts of labeled training data, analyzing the data for correlations and patterns, and then applying those same patterns to predict future states.

Learning, reasoning, and self-correction are the three cognitive skills that AI programming relies on. AI systems operate on trained data, implying the quality of an AI system is as good as its data; they then employ algorithms that discover patterns from huge amounts of information.

1-1 Types of Artificial Intelligence

Artificial Intelligence can be classified into two types: AI based on capability and AI based on functionality, as illustrated by **Figure 2-1.**

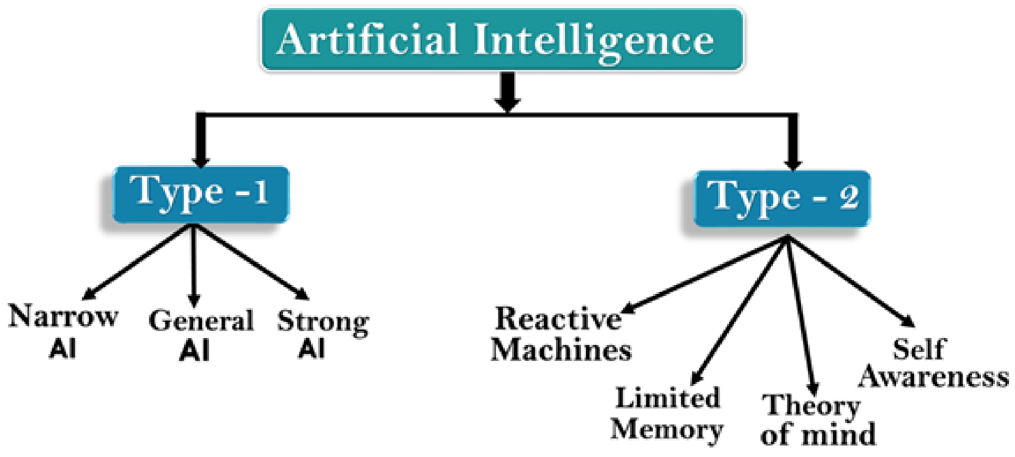


Figure 2-1: Types of Artificial Intelligence [5].

a. AI type-1: Based on Capabilities

The most prevalent Artificial intelligence in the world today is known as **Narrow AI**, it is designed to perform a specific narrow task (e.g. only speech recognition ,playing chess or only driving a car) intelligently.

If narrow AI is pushed beyond its limits, it can fail in unexpected way. It cannot go beyond its field, hence why it is called **weak AI.**

General AI is a type of intelligence that can perform any cognitive task with the same efficiency as a typical human. Currently , there is no such system that falls under General AI , however , it is the long time goal of global researchers to one day achieve an artificial general intelligence (AGI also known as Strong AI).

Super AI is the overall hypothetical system that research aims to achieve, as it is a level at which machine performance surpasses the human mind, and is capable of outperforming it in every task that requires intellectual capabilities. Artificial Superintelligence (ASI) is still a speculative AI concept as its real world development would be ground breaking, although it is predicted to be an after effect of creating general Ai. **Figure 2-2** presents the capability based AI systems.

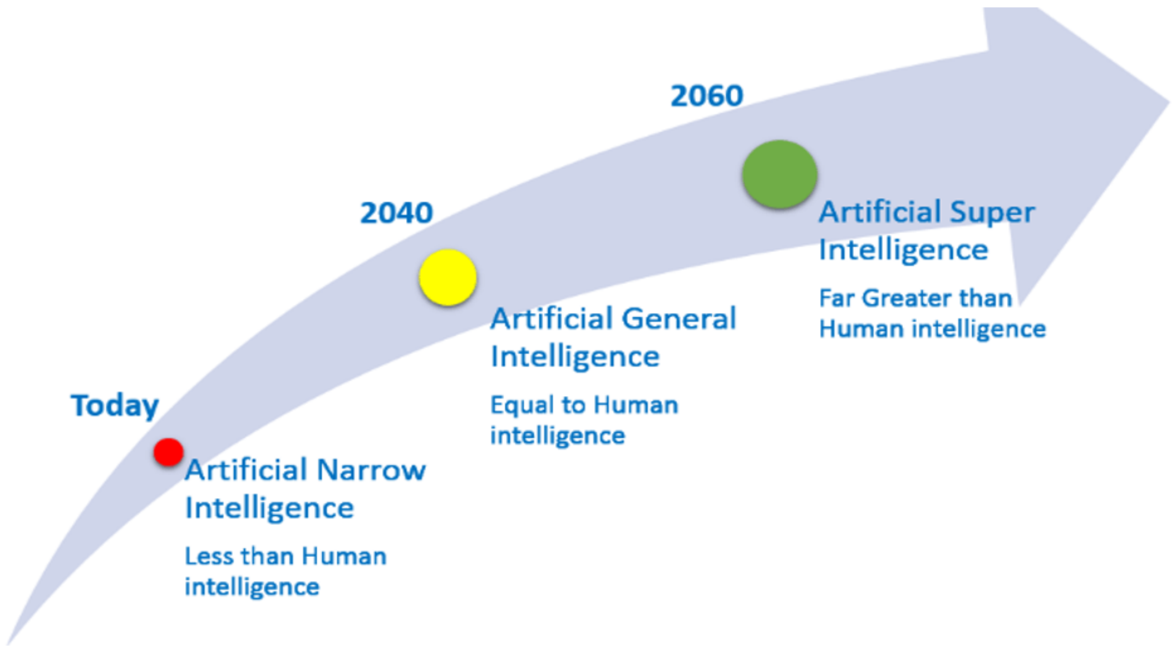

Today
Artificial Narrow Intelligence
Less than Human intelligence
2040
Artificial General Intelligence
Equal to Human intelligence
2060
Artificial Super Intelligence
Far Greater than Human intelligence

Figure 2-2: Capability based AI systems [6].

b. AI type-2: Based on Functionality

1. Reactive Machines

This category of AI includes machines that operate solely on current data, taking only the current situation into account. Reactive AI machines are unable to draw conclusions from data in order to plan their future actions. They can only perform a limited set of pre-defined tasks.

The famous IBM Chess program that defeated world champion Garry Kasparov is an example of Reactive AI.

2. Limited Memory

As the name implies, Limited Memory AI can make better decisions by studying past data from its memory. Such an AI has a temporary or short-lived memory that can be utilized to store previous experiences and thus evaluate future actions.

One of the best examples of Limited Memory systems is self-driving cars. These vehicles can store the most recent speed of nearby vehicles, the distance between vehicles, the speed limit, as well as other information to help them navigate the road.

3. Theory of Mind

The Theory of Mind is a more sophisticated form of Artificial Intelligence. This type of machine is thought to play a significant role in psychology. This type of AI will primarily focus on emotional intelligence in order to better understand human beliefs and thoughts. It should be able to understand human emotions, people, and beliefs, as well as interact socially with humans. This type of AI machine has yet to be developed, but researchers are working hard to improve their chances of success.

4. Self-Awareness

Self-awareness AI is the future of Artificial Intelligence. These machines will be extremely intelligent, with their own consciousness, feelings, and self-awareness. They will be more intelligent than the human mind. Self-Awareness AI does not exist in reality and is only a theoretical concept.

1-2. The Major Branches of Artificial Intelligence

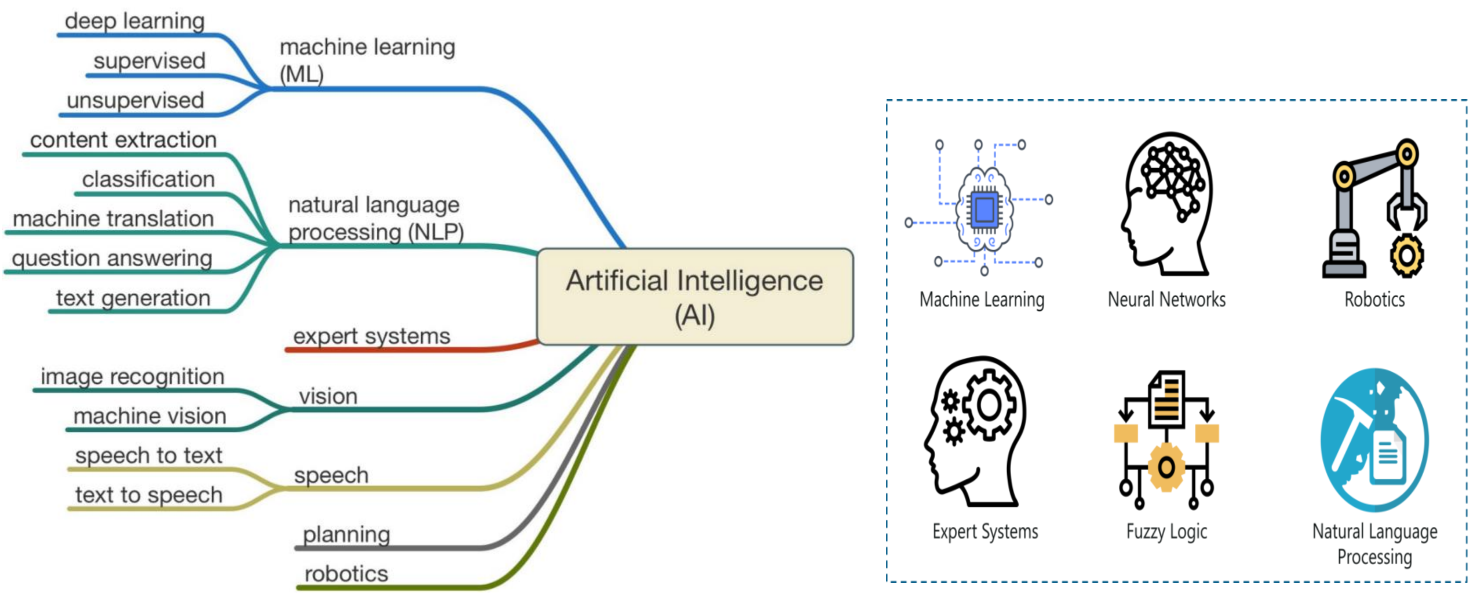


Figure 2-3: The major branches of Artificial Intelligence [7].

a. Natural Language Processing

Natural language processing (NLP) enables computers to communicate with people in their native language while also automating other language-related tasks. Simply put, this is the process of teaching computer systems and machines basic human interactions. A machine receives human sound from interaction and converts it to text format so that it can be easily read and understood. The computer system then converts these texts into components that allow it to understand the human's intent.

b. Expert Systems

An expert system is a computer program that is designed to solve complex problems and make decisions in the same way that a human does. And learns and mimics human decision-making abilities. It accomplishes this by extracting knowledge from its knowledge base using reasoning and inference rules based on user queries.

c. Robotics

Robotics is a technology branch that deals with robots. Robots are programmable machines that can typically perform a series of actions autonomously or semi-autonomously. A robot is comprised of three major important factors:

1. Sensors and actuators allow robots to interact with the physical world.

2. Robots can be programmed.

3. Most robots are autonomous or semi-autonomous.

d. Fuzzy Logic

Fuzzy logic is a computing approach based on the principles of “degrees of truth” instead of the usual modern computer logic i.e. Boolean in nature.

e. Neural Networks

Another area of AI research, neural networks, is inspired by the natural neural network of the human nervous system.

The concept of ANNs is based on the belief that by making the right connections, the workings of the human brain can be imitated using silicon and wires as living neurons and dendrites.

The human brain is made up of 86 billion nerve cells known as neurons. Axons connect them to thousands of other cells. Dendrites accept stimuli from the external environment as well as inputs from sensory organs. These inputs generate electric impulses that travel quickly through the neural network. A neuron can then forward the message to another neuron to handle the problem, or it can ignore it.

ANNs are made up of multiple nodes that mimic biological neurons in the human brain. The neurons are linked together and interact with one another. The nodes can accept input data and perform basic operations on it. The outcome of these operations is communicated to other neurons. Each node's output is referred to as its activation or node value.

Each link has a weight associated with it. ANNs have the ability to learn by changing their weight values.

2. Machine Learning

The concept of machine learning, or the idea that a computer can learn an abstract concept from data and apply it to situations that have not yet been observed, is not new and has been present at least since the 1950s. However, during the past several years, there has been a surge in interest in machine learning (ML) and artificial intelligence, which is being fueled by the enormous and constantly growing amounts of data and computing power as well as the development of better learning algorithms.

Machine learning (ML) is a subfield of Artificial intelligence and it is devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks [8]. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks. There are three machine learning types: **supervised, unsupervised, and reinforcement learning.** As illustrated by **Figure 2-4.**

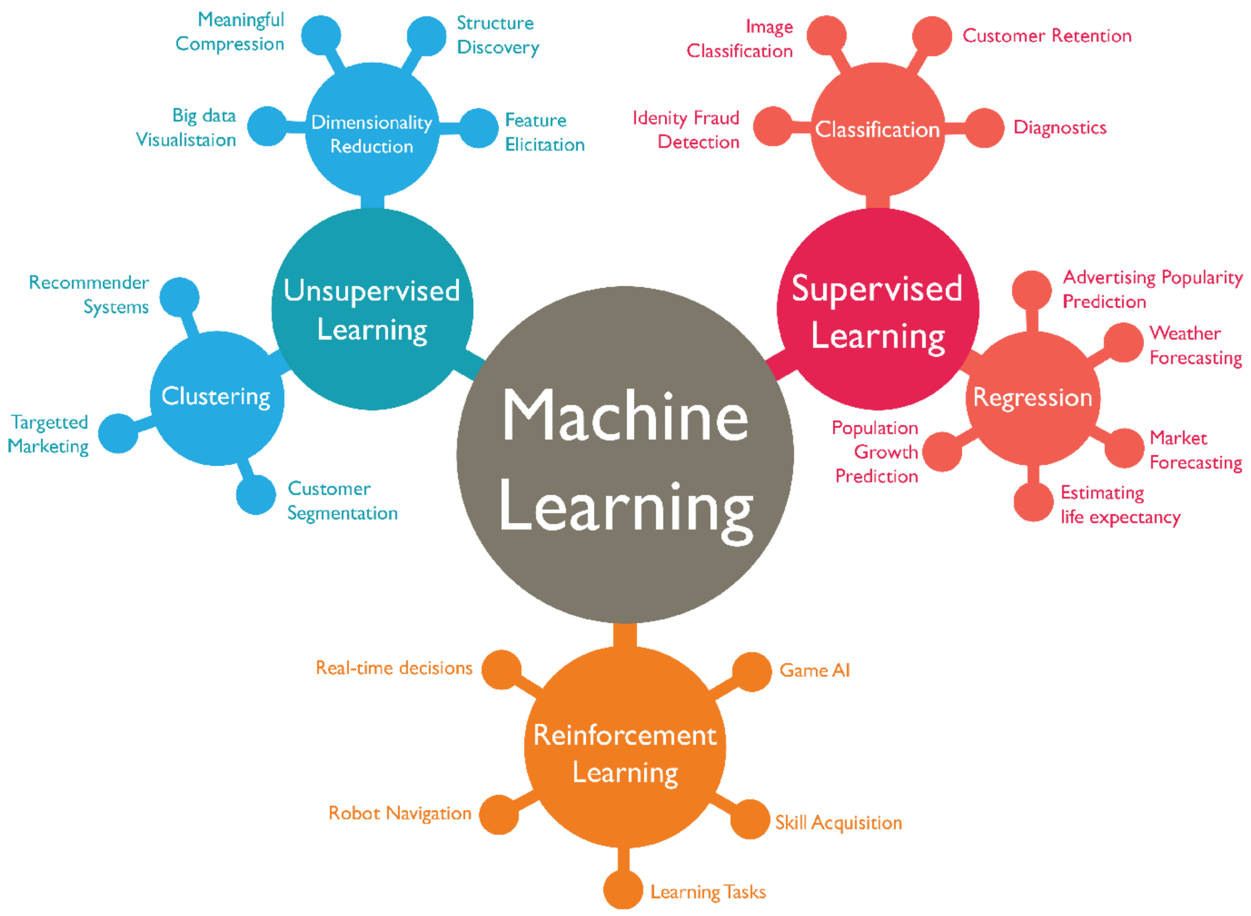


Figure 2-4: The types of machine learning.

2-1. Supervised Learning

Supervised learning allows you to collect data or produce a data output from a previous ML deployment. Supervised learning works in a similar way to how humans learn.

In supervised tasks, we give the computer a training set, which is a collection of labeled data points .A training set is used in supervised learning to teach models to produce the desired output. It includes inputs and correct outputs, allowing the model to learn over time. The algorithm evaluates its accuracy using the loss function and adjusts until the error is sufficiently minimized. Supervised learning is represented in **Figure 2-5**.

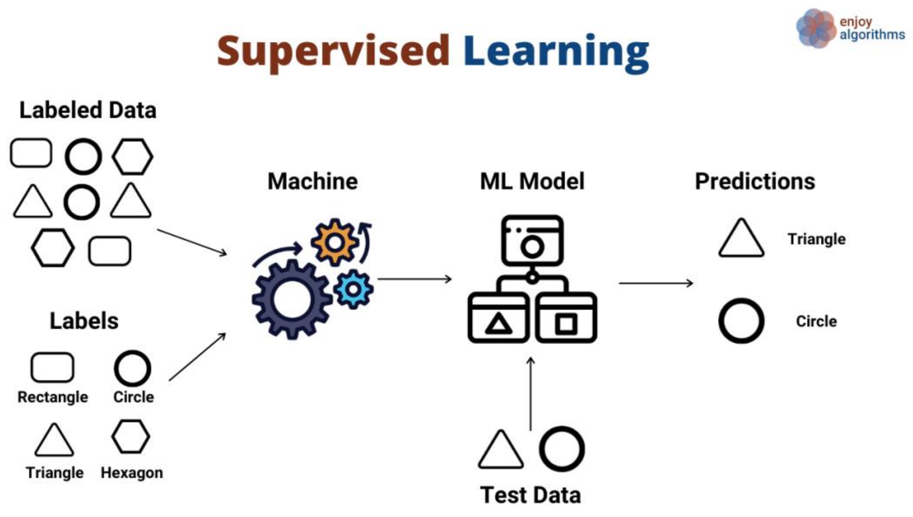


Figure 2-5: Supervised learning [9].

When it comes to data mining, supervised learning can be divided into two types of problems: classification and regression:

- **Classification:** An algorithm is used in classification to accurately assign test data to specific categories. It identifies specific entities within the dataset and tries to draw conclusions about how those entities should be labeled or defined. Linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forests are examples of common classification algorithms.
- **Regression** is a statistical method for determining the relationship between dependent and independent variables. It is commonly used to make projections, such as those for a company's

sales revenue. Popular regression algorithms include linear regression, logistic regression, and polynomial regression.

2-2. Unsupervised Machine Learning

Unsupervised machine learning allows to discover previously unknown patterns in data. With only unlabeled examples, the algorithm attempts to learn some inherent structure to the data. Clustering and dimensionality reduction are two common unsupervised learning tasks. **Figure2-6** illustrates unsupervised learning.

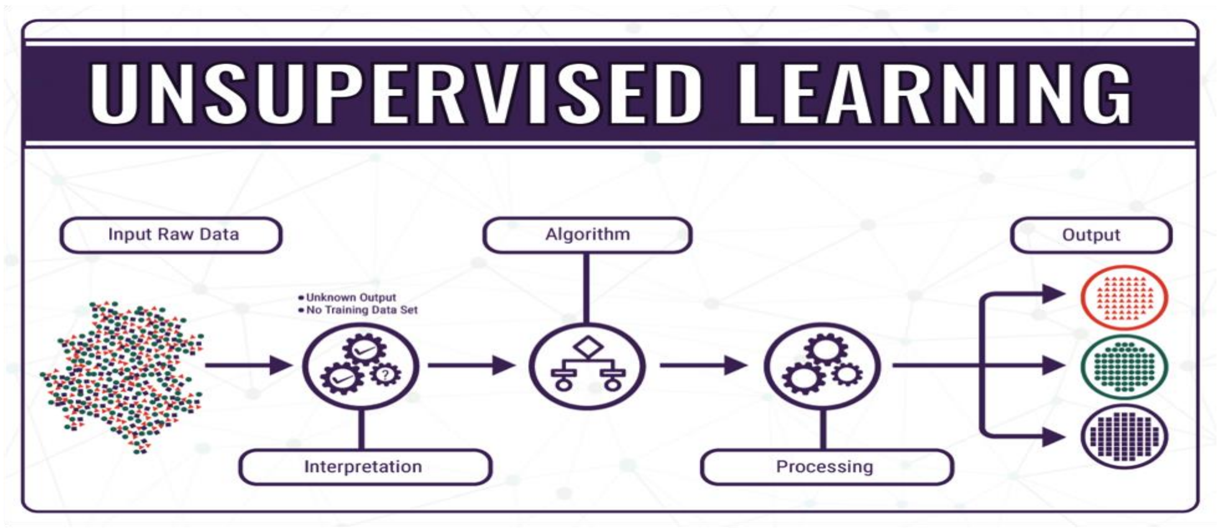


Figure 2-6: Unsupervised learning [9].

Clustering attempts to group data points into meaningful clusters so that elements within a given cluster are similar to one another but different from those in other clusters. Clustering algorithms can be categorized into a few types, specifically exclusive, overlapping, hierarchical, and probabilistic.

Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction. While more data generally yields more accurate results, it can also have an impact on the performance of machine learning algorithms (e.g., overfitting) and make dataset visualization difficult. When the number of features, or dimensions, in a given dataset is too large, dimensionality reduction is used. It reduces the number of data inputs to a manageable number while preserving the dataset's integrity as much as possible. It is commonly used during the preprocessing stage of data, and there are several dimensionality reduction methods available.

2-3. Reinforcement Machine Learning

Reinforcement learning is a machine learning training method that rewards desired behaviors while punishing undesirable ones. A reinforcement learning agent, in general, can perceive and interpret its environment, act, and learn through trial and error. As depicted in **Figure 2-7.**

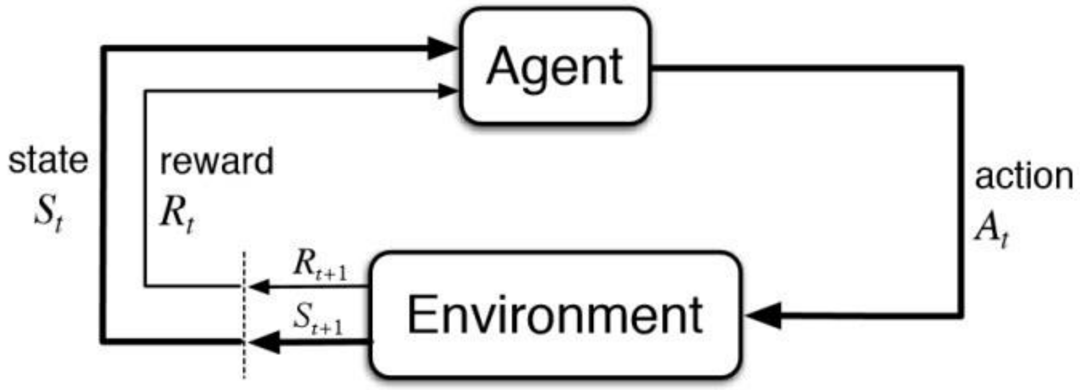


Figure 2-7: Reinforcement ML.

It is up to the model to figure out how to perform the task in order to maximize the reward, beginning with completely random trials and progressing to sophisticated tactics and superhuman abilities. Reinforcement learning is currently the most effective way to hint machine creativity by leveraging the power of search and many trials. Unlike humans, artificial intelligence can learn from thousands of parallel gameplays if a reinforcement learning algorithm is run on a powerful enough computer infrastructure.

3. Artificial Neural Networks (ANNs)

Artificial neural networks are a class of machine learning algorithms that simulate the mechanism of learning in biological organisms. The human nervous system contains cells, which are referred to as neurons. The neurons are connected to one another with the use of axons and dendrites, and the connecting regions between axons and dendrites are referred to as synapses. These connections are illustrated in **Figure 2-8**. The strengths of synaptic connections often change in response to external stimuli. This change is how learning takes place in living organisms. This biological mechanism is simulated in artificial neural networks, which contain computation units that are referred to as neurons [10].

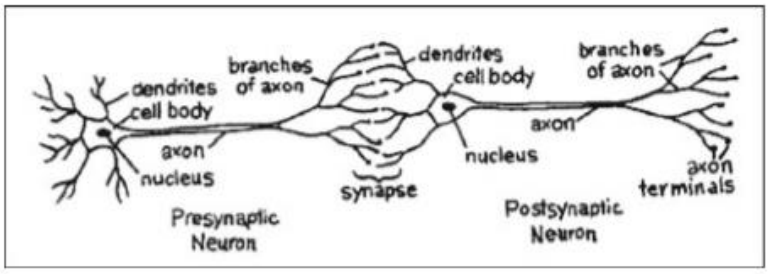


Figure 2-8: Biological neural network [10].

The analogous structure of both biological neurons and artificial neurons is depicted on **Figure 2-9**

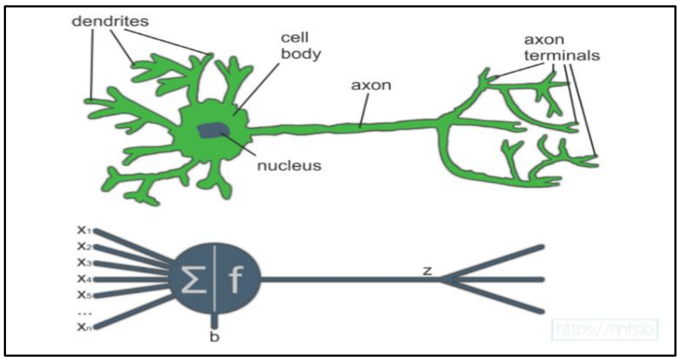


Figure 2-9: Comparison between the structure of a biological neuron and an artificial neuron [11].

3-1. Basic Architecture of Neural Networks

The simplest neural network is referred to as the perceptron, also known as single computational layer. This neural network contains a single input layer and an output node. The basic architecture of the perceptron can be seen in **Figure 2-10.**

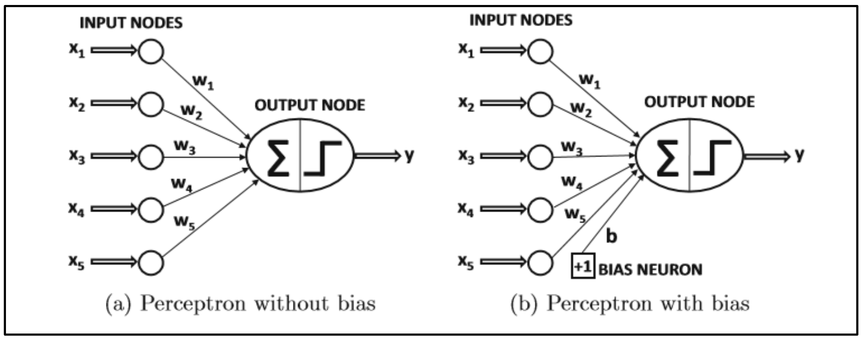


Figure 2-10: The basic architecture of the perceptron.

Neural networks are complex structures composed of artificial neurons that can accept multiple inputs and produce a single output. A Neural Network's primary function is to convert input into meaningful output. A Neural Network typically consists of an input and output layer, as well as one or more hidden layers.

All neurons in a Neural Network influence each other and are thus all connected. The network can recognize and observe every aspect of the dataset at hand, as well as how the various parts of data may or may not be related to one another. This is how Neural Networks can find extremely complex patterns in massive amounts of data.

The flow of information in a Neural Network occurs in two ways:

- **Feedforward Networks**: The signals in this model only travel in one direction, towards the output layer. Feedforward networks have an input layer, a single output layer, and one or more hidden layers. They are commonly employed in pattern recognition.
- **Feedback Networks**: In this model, recurrent or interactive networks process the sequence of inputs using their internal state (memory). Signals can travel in both directions through the network's loops (hidden layer/s). They are commonly employed in time-series and sequential tasks.

3-2. Neural Network Components

Figure 2-11 represents the architecture of neural networks.

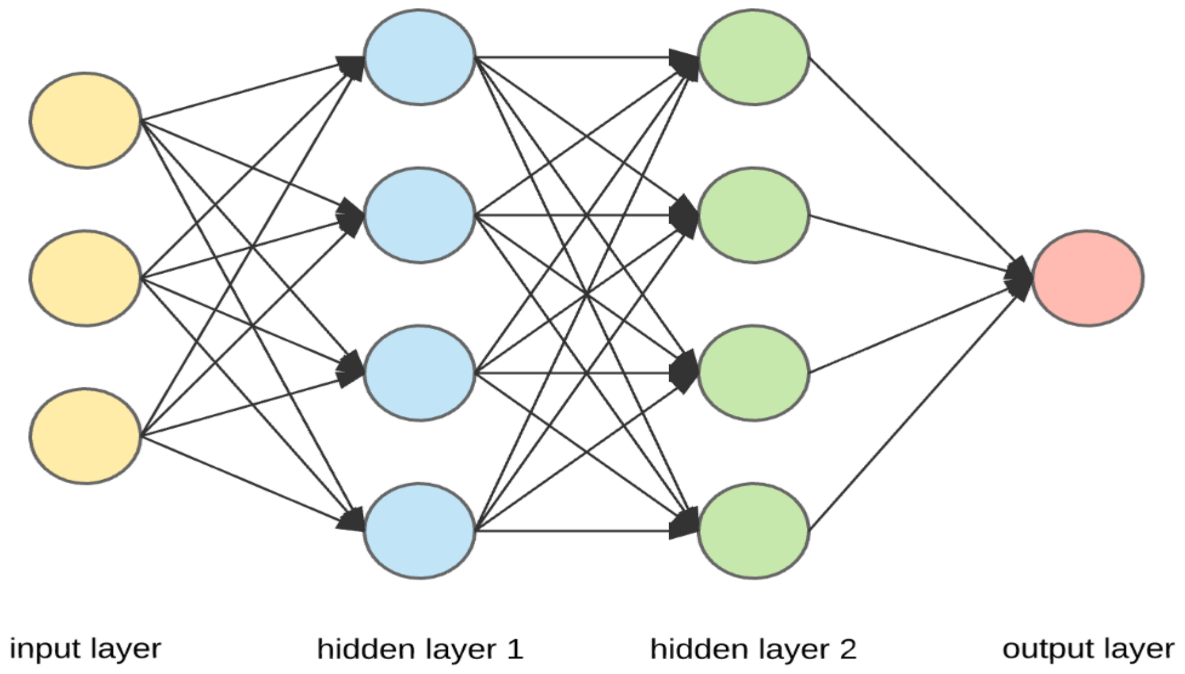


Figure 2-11: The architecture of neural networks.

a. Input layer

In **Figure 2-11**, the input layer is the yellow outermost layer. A neuron is the basic building block of a neural network. They receive input from a remote source or from other nodes. Each node is connected to another node in the next layer, and each connection has a weight. Weights are assigned to neurons based on their relative importance in comparison to other inputs.

When all of the node values from the input layer are multiplied and summarized (along with their weight), a value for the first hidden layer is generated. The blue layer has a predefined "activation" function that determines whether or not this node will be "activated" and how "active" it will be based on the summarized value.

b. Hidden layer

The hidden layer is the layer or layers that are hidden between the input and output layers. The hidden layer is so named because it is always hidden from the outside world.

The hidden layers of a Neural Network are where the majority of the computation occurs. As a result, the hidden layer takes all of the inputs from the input layer and runs the necessary calculations to produce a result. This result is then sent to the output layer, where the user can see the outcome of the computation.

Hidden layers allow a neural network's function to be broken down into specific data transformations. Each hidden layer function is tailored to produce a specific result. For example, a hidden layer function that identifies human eyes and ears may be used by subsequent layers to identify faces in images. While the functions to identify eyes alone are insufficient to recognize objects independently, they can work together within a neural network.

Hidden layers are common in neural networks, but their application and architecture vary widely. As previously stated, hidden layers can be distinguished by their functional characteristics. In a CNN used for object recognition, for example, a hidden layer used to identify wheels cannot identify a car on its own; however, when combined with additional layers used to identify windows, a large metallic body, and headlights, the neural network can then make predictions and identify possible cars within visual data [12]. **Figure 2-12** depicts an artificial neuron.

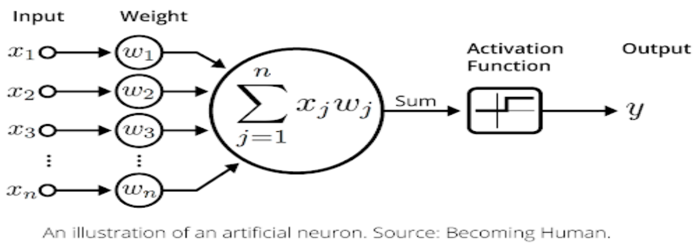


Figure 2-12: Illustration of an artificial neuron.

3-3. Neural network Algorithms

The learning (or training) process in a Neural Network begins by dividing the data into three distinct sets:

- **Training Dataset:** The initial data used to train machine learning models is known as training data (or a training dataset).
  Machine learning algorithms are fed training datasets to teach them how to make predictions or perform a desired task.
- **Validation Dataset:** A validation dataset is a sample of data from your model's training that is used to estimate model skill while tuning the model's hyperparameters.
- **Test Dataset:** Data that has been specifically identified for use in tests, typically of a computer program, is referred to as test data. Some data can be used to confirm that a given set of input to a given function produces the expected result.

After the data has been segmented into these three parts, Neural Network algorithms are used to train the Neural Network. The optimization procedure is used to facilitate the training process in a Neural Network, and the algorithm used is known as the optimizer. There are various types of optimization algorithms, each with its own set of characteristics and features such as memory requirements, numerical precision, and processing speed.

3-4. Backpropagation

The training process in a single-layer neural network is relatively simple because the error (or loss function) can be computed as a direct function of the weights, allowing for easy gradient computation. The problem with multi-layer networks is that the loss is a complicated composition function of the weights in previous layers. The backpropagation algorithm is used to compute the gradient of a composition function. It employs the differential calculus chain rule to compute error gradients as summations of local-gradient products over the various paths from a node to the output.

Despite the fact that this summation has an exponential number of components (paths), it can be efficiently computed using dynamic programming. Dynamic programming is used directly in the backpropagation algorithm. It has two major phases, known as the forward and backward phases, respectively. The forward phase must compute the output values and local derivatives at each node, and the backward phase must accumulate the products of these local values along all paths from the node to the output.

1. **Forward phase**: The inputs for a training instance are fed into the neural network during this phase. This causes a forward cascade of computations across the layers to be performed using the

current set of weights. The predicted output is compared to the training instance, and the derivative of the loss function with respect to the output is calculated. This loss' derivative must now be computed with respect to the weights in all layers in the backwards phase [10].

2. **Backward phase**: The primary goal of the backward phase is to learn the gradient of the loss function with respect to the different weights using differential calculus' chain rule. The weights are updated using these gradients. Because these gradients are learned backwards, beginning with the output node, this learning process is referred to as the backward phase [10].

Backpropagation Algorithm Set all weights to small, random numbers [13].

• For each training example, do

1. Input the training example and compute the results.
2. For each output unit k:

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h:

$$\delta_k \leftarrow o_k(1 - o_k) \sum_{koutputs} w_{h,k}\delta_k$$

4. Update each network weight:

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

Where: $$\Delta w_{i,j} = \eta\delta_j x_{i,j}$$

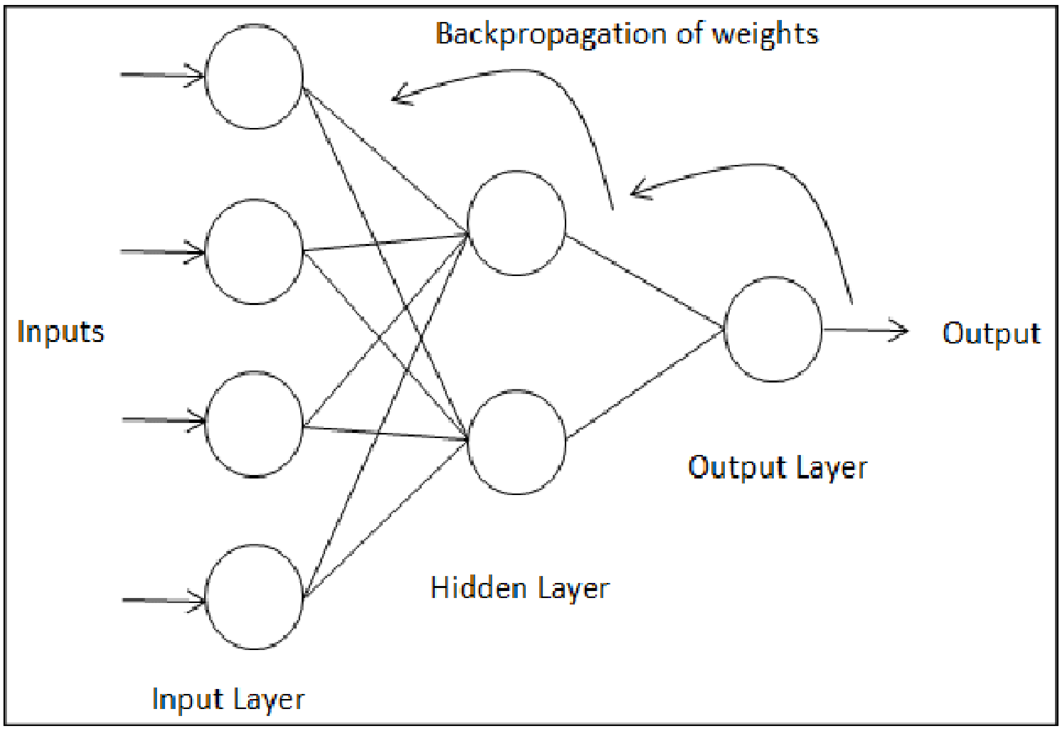


Figure 2-13: Backpropagation illustration [14].

Backpropagation is illustrated in **Figure 2-13.**

3-5. Mean squared error MSE

The Mean Squared Error (MSE) is perhaps the most basic and widely used loss function, and it is frequently taught in introductory Machine Learning courses. To compute the MSE, divide the difference between your model's predictions and the ground truth by two, square it, and average it over the entire dataset.

Because we are always squaring the errors, the MSE will never be negative. The MSE is mathematically defined as it is shown in **Figure 2-14**.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Figure 2-14: MSE equation.

4. The project hardware

In this project, we will use the field programmable gate array FPGA integrated circuit to embed a pattern recognition application, which will control a stepper motor.

4-1. The FPGA

A Field Programmable Gate Array (FPGA) is an integrated circuit made up of internal hardware blocks with user-programmable interconnects that allow it to be customized for a specific application's operation. They are easily reprogrammable, allowing an FPGA to accommodate design changes or even support a new application during the part's lifetime.

The FPGA evolved from earlier devices like programmable read-only memories (PROMs) and programmable logic devices (PLDs). In the late 1980s, the concept of FPGA was created through an experiment suggested by Steve Casselman. His goal was to create a computing device with over 600,000 reprogrammable gates. FPGA stores its configuration data in a reprogrammable medium such as static RAM (SRAM) or flash memory. Intel, Lattice Semiconductor, Microchip Technology, and Microsemi are among the FPGA manufacturers [15].

The FPGA architecture is typically configured using a language similar to that used for ASICs (Application Specific Integrated Circuit), namely HDL (Hardware Description Language).

The typical FPGA architecture (**Figure 2-15**) is made up of three kinds of modules. I/O blocks or Pads, Switch Matrix/Interconnection Wires, and Configurable logic blocks (CLB). The fundamental FPGA architecture consists of two-dimensional arrays of logic blocks with the ability for the user to configure the connectivity between the logic blocks. The following are the functions of an FPGA architectural module:

- The CLB (Configurable Logic Block) consists of digital logic, inputs, and outputs. It carries out the user logic.
- Interconnects provide guidance between logic blocks in order to implement the user logic.
- Switch matrix provides switching between interconnects based on the logic.
- I/O Pads allow the outside world to communicate with various applications.

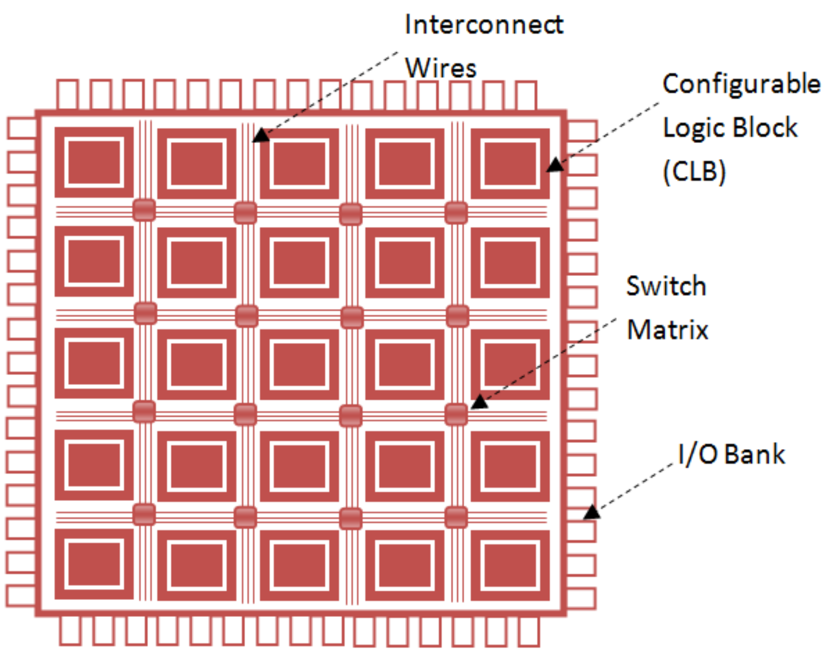


Figure 2-15: The FPGA architecture [16].

MUX (Multiplexer), D flip flop, and LUT are all part of the logic block. The LUT implements combinational logical functions; the MUX handles selection logic; and the D flip flop stores the LUT output.

The Look Up Table-based function generator is the FPGA's fundamental building block. After experiments, the number of inputs to the LUT ranges from 3, 4, 6, and even 8. With the implementation of two function generators, we now have adaptive LUTs that provide two outputs per single LUT. An illustration of configurable logic block is shown in **Figure 2-16**.

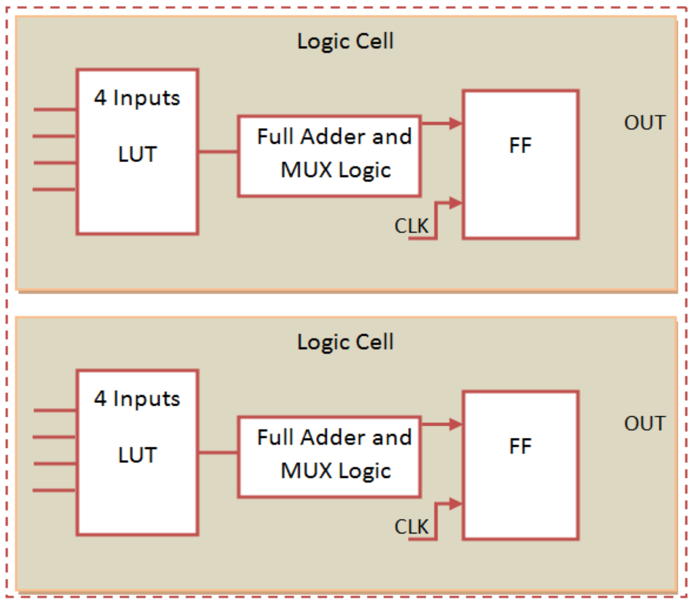


Figure 2-16: Configurable Logic Block.

The 4 input Lookup table LUT is used to implement one of the following features: Combinational Logic design, Distributed RAM, Shift Register. The current FPGA is made up of millions of configurable logic blocks.

4-1-1. FPGA types:

FPGAs are classified into three types based on their applications: low-end FPGAs, mid-range FPGAs, and high-end FPGAs [17].

a. Low End FPGAs

These FPGAs are intended to have low power consumption, low logic density, and low complexity per chip. Low-end FPGAs include Altera's Cyclone family, Xilinx's Spartan family, Microsemi's Fusion family, and Lattice semiconductor's Mach XO/ICE40 [17].

b. Mid Range FPGAs

These FPGAs are the best compromise between low-end and high-end FPGAs, and they are designed to strike a balance between performance and cost. Arria from Altera, Artix-7/Kintex-7 series from Xlinix, IGL002 from Microsemi, and ECP3 and ECP5 series from Lattice semiconductor are examples of mid-range FPGAs [17].

c. High End FPGAs

These FPGAs are designed for high logic density and performance. High-end FPGAs include Altera's Stratix family, Xilinx's Virtex family, Achronix's Speedster 22i family, and Microsemi's ProASIC3 family [17].

4-2. FPGA Applications

The wide range of applications in which FPGAs can operate (**Figure 2-17**) has made their growth significant over the past decade. Examples of the applications of FPGAS are Digital signal processing, bioinformatics, device controllers, software-defined radio, random logic, ASIC prototyping, medical imaging, computer hardware emulation, integrating multiple SPLDs, voice recognition, cryptography, filtering, communication encoding, and many more.

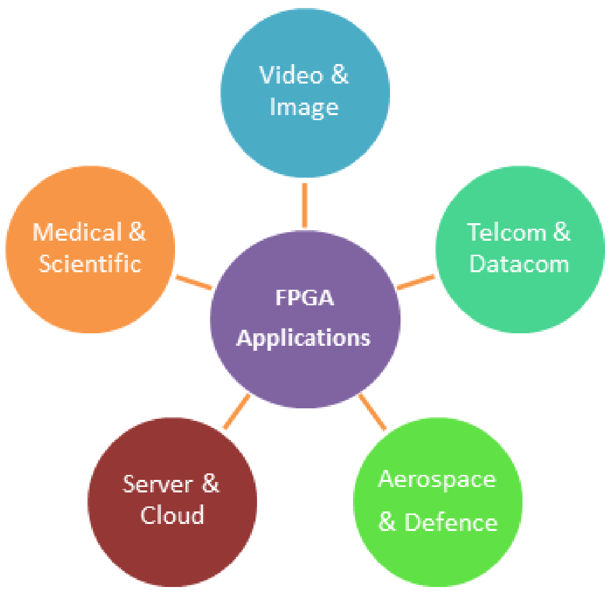


Figure 2-17: FPGA Applications [18].

a. Communications, Software Defined Radio (SDR)

Software Defined Radio/Network and other complex algorithms such as FFT must be implemented in FPGA in order to be used in a Hard Real-Time environment. Traditionally, a radio consisted of an antenna for receiving and transmitting a signal, as well as hardware for processing that signal, filtering it, modifying its frequency, and so on. This hardware was not capable of significantly altering the functionality for which it was designed. Much of this functionality is now transferred to an electronic device, which is frequently an FPGA, and the analog component can be limited to an antenna and ADC and DAC converters. The main advantage of this type of radio is that its functionality is defined by the software design, making modification or updating simple and requiring no hardware replacement. In some cases, ADC and DAC are also integrated into the FPGA chip.

b. Artificial vision systems

In today's world, an increasing number of devices have an artificial vision system. Video surveillance cameras, robots, and other similar devices are examples of this. Many of these devices require a system to know their location, recognize objects in their surroundings, recognize people's faces, and act and interact appropriately with them. This feature necessitates dealing with massive amounts of images and processing them in real time to detect objects, recognize faces, and so on.

c. Medical imaging systems

FPGAs are increasingly being used to treat biomedical images obtained through PET processes, CT scans, X-rays, three-dimensional images, and so on. These medical vision systems growingly require higher resolution and greater processing capacity, and many must be developed in real time, so the benefits of frequency FPGAs and parallel processing are well suited to these requirements.

d. Encryption decryption and cryptography

massive computing parallelism, the ability to configure the computational units to the bit-width required, and low latency are the primary reasons why FPGAs are used in encrypting/decrypting and Post-quantum cryptography [19].

e. Radio astronomy

Radio astronomy is the science that studies the phenomena that occur in space by collecting electromagnetic radiation from it. Similar to previous applications, it necessitates the processing of a large amount of data in order for the FPGA to reach its full potential.

f. Speech recognition

Speech recognition is a technique used in security, information retrieval systems, and other applications, and its range of applications is expected to expand in the future. When comparing a person's voice to previously stored patterns, the FPGA is very efficient in this context.

g. Aeronautics and defense

In addition to the other applications FPGAs are used in a wide range of aeronautical and defense applications due to the benefits they provide.

h. Data Center / Cloud

The internet of things (IoT) and big data in general are causing an exponential increase in the amount of data acquired and processed, which, combined with computational analysis of the same using deep learning techniques of multiple operations parallel / concurrent, is causing a high demand for low-latency, flexible, and secure computational capacity, which cannot be met by adding more servers / blades due to the insane increase in space cost. Under this scenario, FPGAs are rapidly entering the Data Center world due to their capacity for computational acceleration, configuration flexibility, and the security that hardware provides against software [19].

i. Control engineering

The ability to implement an FPGA-based controller as a hard Real-Time system, that is capable of responding to any time-critical changes inside the controlling environment, in a calculated deterministic time. Another feature is the ability to reconfigure the FPGA during run-time, allowing adaptation to a changing environment by selecting the best fitting controller algorithm while reducing the required logic resources and deployment time.

4-3. The DE2-115 Development and Education board

The DE2 series has consistently been at the forefront of educational development boards by distinguishing itself with a plethora of interfaces to accommodate a wide range of application requirements, extending its dominance and success.

The DE2-115 provides an optimal balance of low cost, low power, and a comprehensive set of logic, memory, and DSP capabilities. The Cyclone EP4CE115 device, which is based on the DE2-115, has 114,480 logic elements (LEs), which is the largest of the Cyclone IV E series, up to 3.9-Mbits of RAM, and 266 multipliers.

Figure 2-18 depicts a photograph of the DE2-115 board. It shows the layout of the board and where the connectors and key components are located.

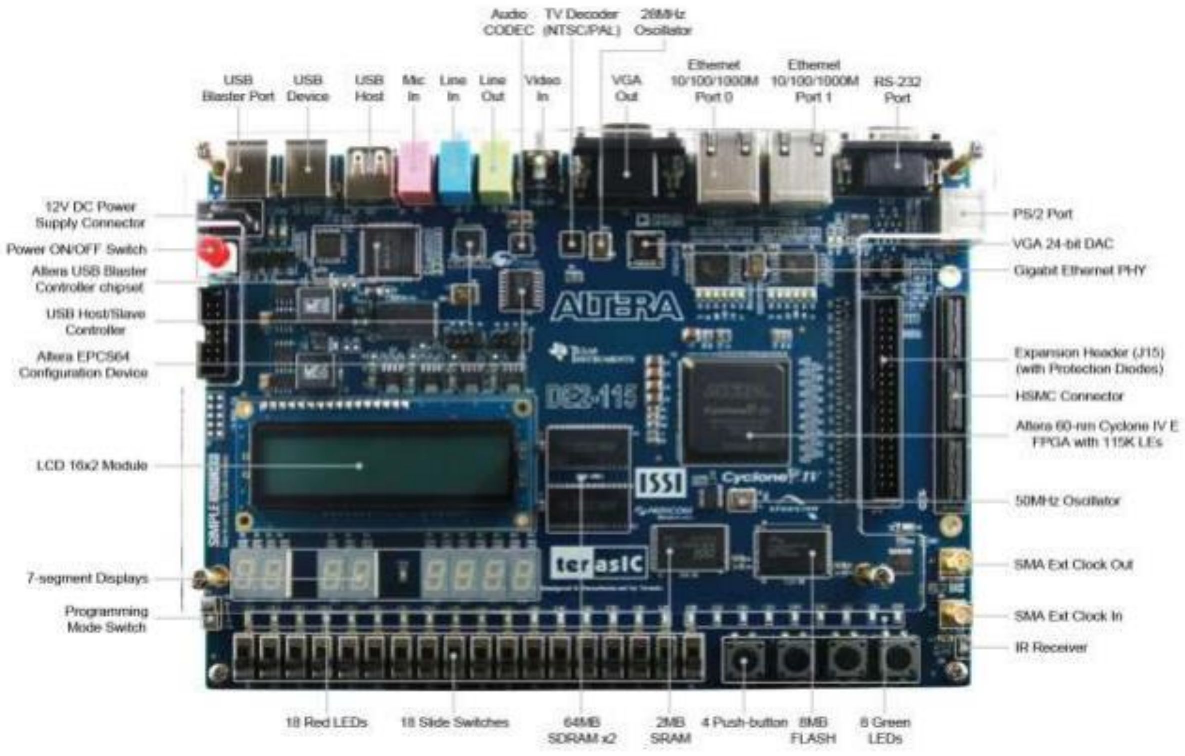


Figure 2-18: The DE2-115 board (top view) [20].

5. The 28BYJ-48 – 5V Stepper Motor

The 28BYJ-48, **Figure 2-19,** is a small stepper motor suitable for a large range of applications [21].

Figure 2-19: The 28BYJ-48 stepper motor [21].

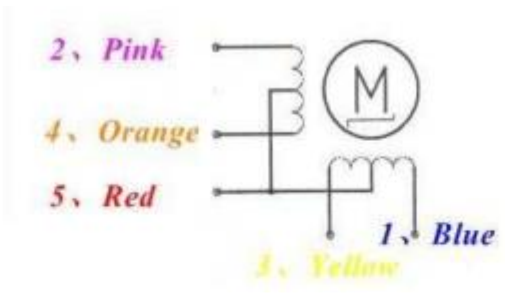


Figure 2-20: The 28BYJ-48 unipolar internal circuitry structure [21].

6. The 4 Phase ULN2003 Stepper Motor Driver PCB

The ULN2003 stepper motor driver PCB connects the microcontroller to a stepper motor directly. The PCB has four inputs for the microcontroller, a power supply connection for the stepper motor voltage, an ON/OFF jumper, a direct connect stepper motor header, and four LEDs to indicate stepping state [22].

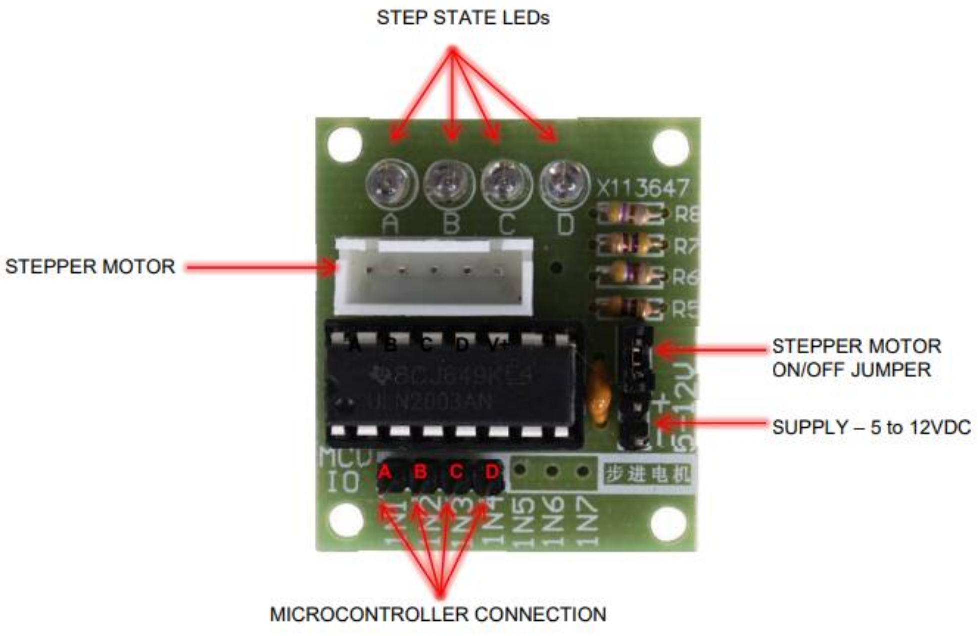


Figure 2-21: The stepper motor driver [22].

Chapter 03

Design and implementation

This chapter examines the detailed design and implementation of the pattern recognizer system and the underlying logic that connects all of its components, as well as the results obtained and a conclusion of the findings.

1. System Overview

The synthesized system using Quartus II software version 13.0 on the DE2-115 FPGA board is a hardware-based pattern recognizer circuit composed of eight main units. A pattern recognizer control unit that manages the operation of the ANN, a floating-point ALU unit designed with megafunctions, an SRAM unit that stores the ANN weights, a linear feedback shift register (LFSR) that generates random initial values for the weights, a display control unit, the built-in LCD module, the ANN block and a motor circuit, **Figure 3-1**.This pattern recognizer is trained to identify 16 different letters that are: A, C, D, F, H, I, J, L, N, O, P, T, U, X, Y, Z and takes its inputs from the 16 toggle switches that populate the pixels of two 4×4 LCD grids alternatively based on the status of switches SW1 and SW0. The output generated by the system is an array of 16 values ranging between 0.0 and 1.0 indicating the percentage of each class occurrence. The highest value specifies the recognized pattern. **Figure 3-2** shows the real picture of the system. When the program is loaded to the board, the pattern recognizer control unit starts training the ANN by uploading the dataset inputs. The weights are updated continuously and stored in the SRAM memory block. Once finished, the green LED number 8 lights up to signal a ready ANN for use. It can be tested by entering any pattern and pushing KEY [3] to trigger the running phase that requires massive computations in the ALU to perform the recognition task. When completed, the display control unit adjusts the LCD to output the result and the motor receives the appropriate command to function. **Figure 3-3** illustrates the working of the system.

The algorithm chosen in this project is a feed forward neural network, also known as a multilayer perceptron with a sigmoid activation function. It is trained with backpropagation technique using a database that consists of a set of 4×4 matrices with values of 1s and 0s representing the pattern shape, as shown in **Figure 3-4.**

Since the selected signals are of float type, all operations are carried out in single precision floating-point format. The advantage of using floating-point numbers over fixed-point numbers is that they can cover a much wider range of values. The radix point is always at the same location in fixed-point number representation. Even though, the convention simplifies and saves memory, it limits the magnitude and precision. A relocatable radix point is desirable in situations that require a wide range of numbers or high resolution. It allows for a reliable implementation of a hardware-based neural network that can recognize interesting features from the set of inputs and produce accurate results compared to when using fixed-point format, which is prone to errors.

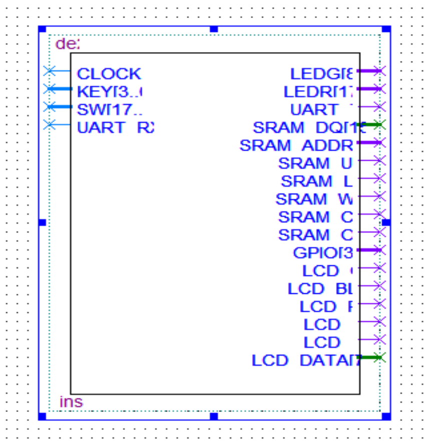


Figure 3-1: The block diagram representation of the system.

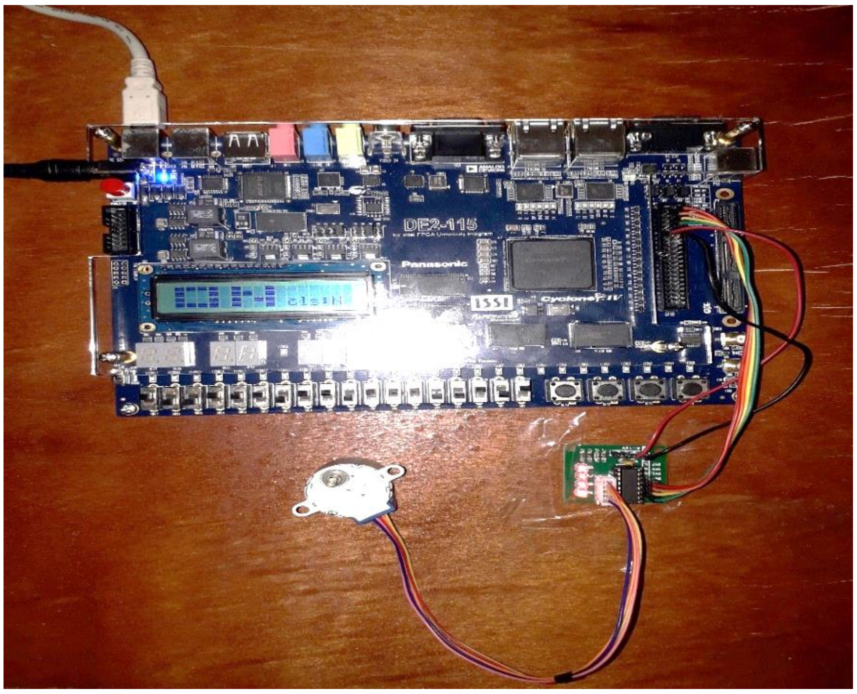


Figure 3-2: Real picture of the system

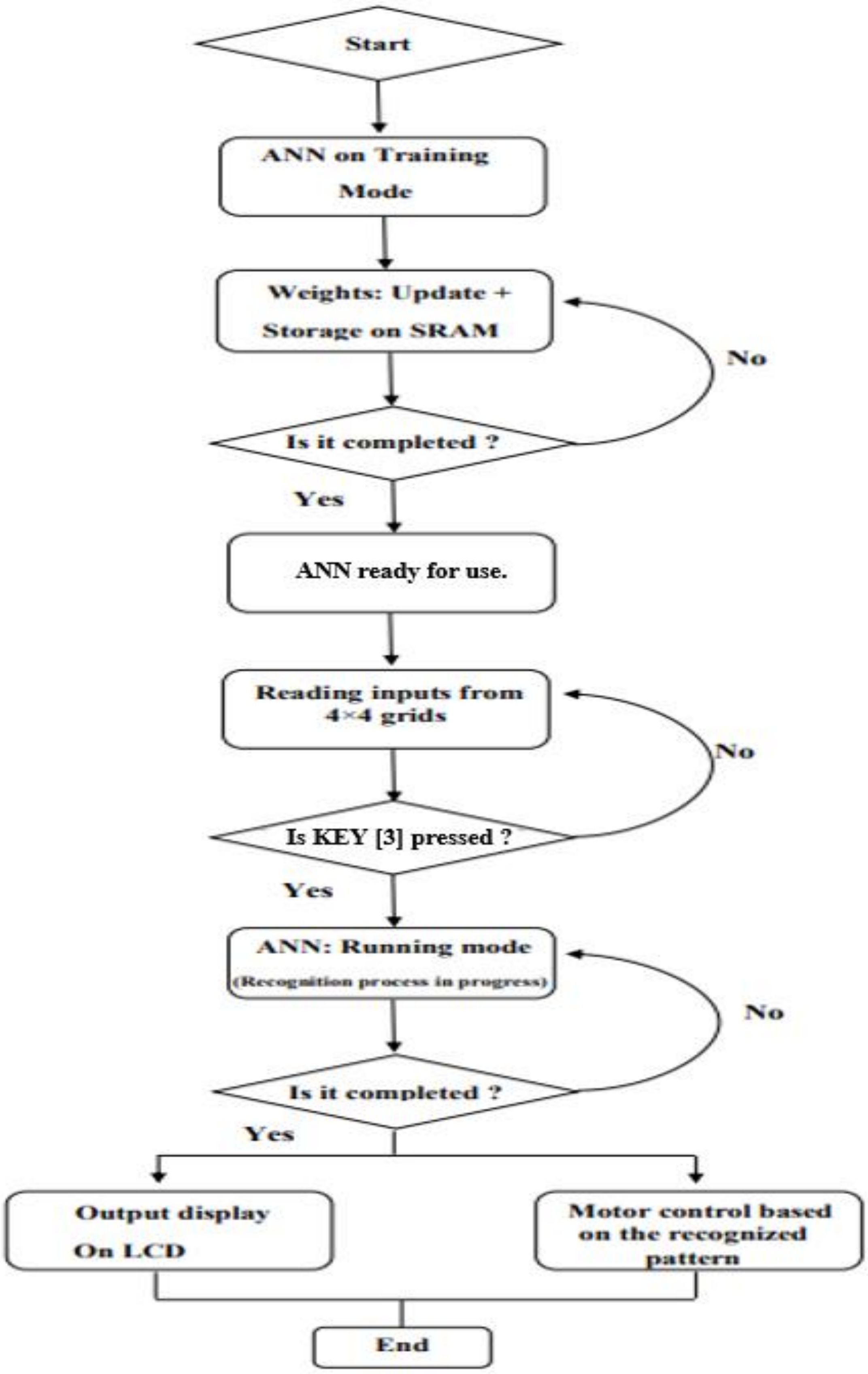
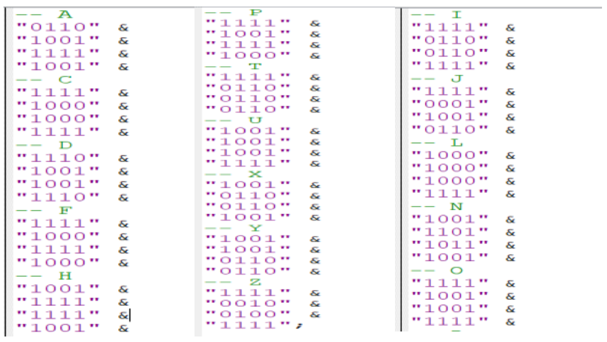


Figure 3-3: The workflow of the system.



```
-- A
"0110"  &
"1001"  &
"1111"  &
"1001"  &
-- C
"1111"  &
"1000"  &
"1000"  &
"1111"  &
-- D
"1110"  &
"1001"  &
"1001"  &
"1110"  &
-- F
"1111"  &
"1000"  &
"1111"  &
"1000"  &
-- H
"1001"  &
"1111"  &
"1111"  &
"1001"  &
```

```
-- P
"1111"  &
"1001"  &
"1111"  &
"1000"  &
-- T
"1111"  &
"0110"  &
"0110"  &
"0110"  &
-- U
"1001"  &
"1001"  &
"1001"  &
"1111"  &
-- X
"1001"  &
"0110"  &
"0110"  &
"1001"  &
-- Y
"1001"  &
"1001"  &
"0110"  &
"0110"  &
-- Z
"1111"  &
"0010"  &
"0100"  &
"1111" ;
```

```
-- I
"1111"  &
"0110"  &
"0110"  &
"1111"  &
-- J
"1111"  &
"0001"  &
"1001"  &
"0110"  &
-- L
"1000"  &
"1000"  &
"1000"  &
"1111"  &
-- N
"1001"  &
"1101"  &
"1011"  &
"1001"  &
-- O
"1111"  &
"1001"  &
"1001"  &
"1111"  &
```

Figure 3-4: The training dataset

2. System Components

2.1 PR Component

The pattern recognizer component is a control unit; designed using finite state machines**.** It handles the running and learning mode of the ANN. It has many input ports such as the asynchronous reset that is assigned the value of KEY [0], the 50 MHz clock available on the board, the training dataset, and the training classes etc. Likewise, it has many output ports such as the output of the recognition process, the ready signal, the ann_alpha, the ann_inputs that represent the input data converted to float type, the ann_targets etc, as illustrated in **Figure 3-5.** It utilizes ALTFP_COMPARE megafunction to carry some needed comparison operations.

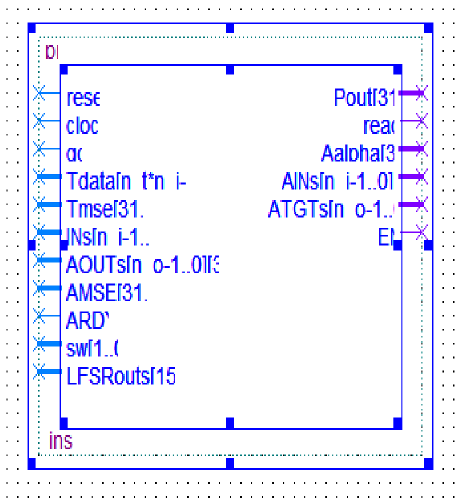


Figure 3-5: Pattern recognizer block diagram representation

Its FSM consists of eleven states that are:

1. Init
2. Train
3. Train_validate
4. Train_validate_wait
5. Train_validate_complete
6. Run
7. Run_complete
8. Run_validate
9. Run_validate_wait
10. Run_validate_complete
11. Idle

Where in each state, a certain set of operations are executed:

Init: In this state, some signals are initialized to zero e.g. the ready output port, as an indication that the pattern recognizer is not ready. The next state is set to train.

Train: The training phase, it converts progressively each value in the corresponding class of the dataset to float type, since the ANN works with floating-point numbers, then the appropriate bit in ann_targets signal is set. These latter are sent to the ANN in order to begin the learning process. The next state is set to the Train_validate.

Train_validate: After the completion of instructions execution in the ANN's learning phase, the floating-point comparator is enabled. It compares between the training_mse and the ann_mse to evaluate whether the error is reduced sufficiently. The next state is set train_validate_wait.

Train_validate_wait: It introduces a necessary delay in the FSM. It then, goes to the following state Train_validate_complete.

Train_validate_complete: In this state, we use a condition statement to check if the ANN has been trained with all of the 16th classes in the dataset; if not, we increment the pointer that indicates the current class it is being trained on. After which we disable the comparator and examine its output. For each class we verify if the output is zero, implying that the training_mse is still less than the ann_mse; we continue the training until the output of the comparator becomes "1" to consider it learned. We therefore, increment the counter of successfully learned classes. Following that, we check if the counter has reached the total number of available classes; if not, the state remains in the training phase; otherwise, the PR block outputs a ready signal to confirm that the user can test the system, and the next state is set to run.

Run: The toggle switches inputs are converted to float type to be transferred to the ANN block. The ANN mode is set to run**.** The next state is run_complete.

Run_complete: At this stage, the comparator is enabled to start comparison between the 16 generated outputs to determine the highest value. The following state is set to run_validate.

Run_validate: We insert the comparator's inputs, to start the comparison between the output values. The following state is run_validate_wait, which accomplishes the same task as Train_validate_wait. The state then goes to Run_validate_complete.

Run_validate_complete: It compares between the 16 outputs and determines the highest one. Once finished it goes to idle state.

Idle: We keep checking if KEY [3] is pressed, to set the ANN mode to run. The FSM diagram of the pattern recognizer component is shown in **Figure 3-6**.

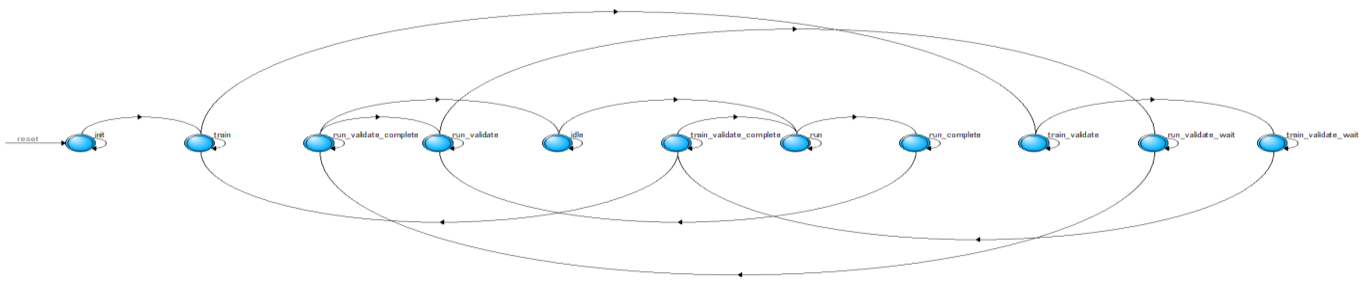


Figure 3-6: The FSM diagram of the pattern recognizer.

2.2 ANN Component

The artificial neural network is designed to be a generic unit in order to introduce flexibility into its structure; in other words, the number of perceptrons in each layer can be reconfigured so that it can be utilized in other applications. The architecture of the ANN consists of 16 perceptrons in the input layer, 32 in the hidden layer, and 16 in the output layer, plus a bias connection for each perceptron that is initially set to a value of one. Because of this architecture, a total of 1072 weights were produced and needed to be stored in memory; hence, we used an external SRAM with relatively cheap storage space to make efficient use of resources. It is considered as the central element in the design since it represents the processing unit that achieves the pattern recognition task by employing an external floating-point ALU to perform the necessary computations.

The ANN takes some of its input ports from the PR block; which are the targets, the toggle switches inputs converted to float type, the ann_mode and the ann_alpha; a constant set to a value of float half that is represented as "00111111000000000000000000000000" in the IEEE-754 32-bit floating point format. It also has the 50 MHz clock, the KEY [0] as an asynchronous reset, the output of the float ALU unit, the SRAM ready signal and SRAM output, and lastly the inputs from the LFSR; which are the generated pseudo-random numbers. Its output ports include a ready signal, a mean squared error (MSE) signal, the float ALU inputs A and B, the SRAM input and address, and the 16 outputs of the recognition process. **Figure 3-7** illustrates its block diagram representation.

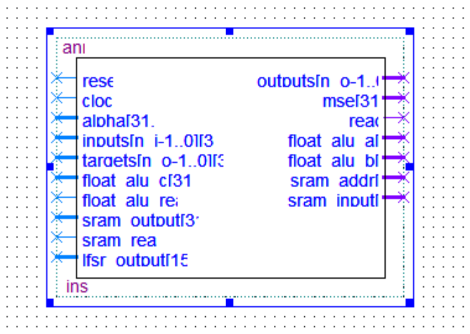


Figure 3-7: Block diagram representation of the Artificial Neural Network.

It has three modes of operation, **Figure 3.8**: idle, learn, and run, which are selected one at a time by the controlling unit (PR block) depending on certain conditions. It executes a specific set of instructions in each of these modes, which is implemented as a complex FSM involving 92 states that can be classified into four major states (phases) composed of many substates that represent the instructions to be done.

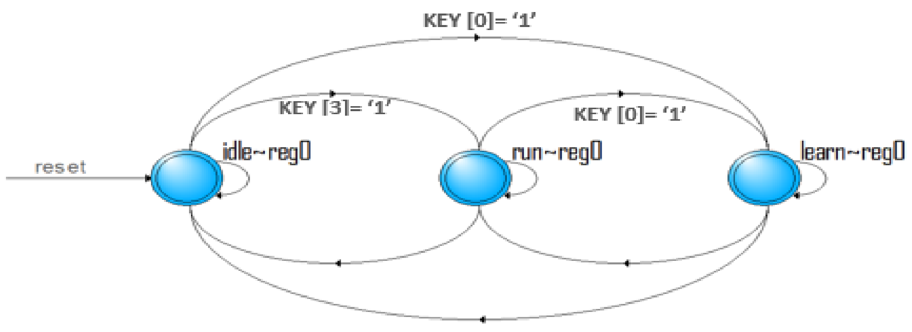


Figure 3-8: The ANN modes FSM diagram.

The four major states of the FSM are:

1) The Initialization Phase

In this phase, some variables, such as the SRAM address and the ready signal, are set to zero, and the mse output port is set to float zero. In order to start the process of storing the random initialized values of the weights, the SRAM address gets loaded with the first value that is zero, to store the first SRAM input. All of the randomized weights that are the SRAM inputs share the same structure that utilizes the 16 outputs of the LFSR, **Figure 3.9**. The SRAM mode is set to write mode.

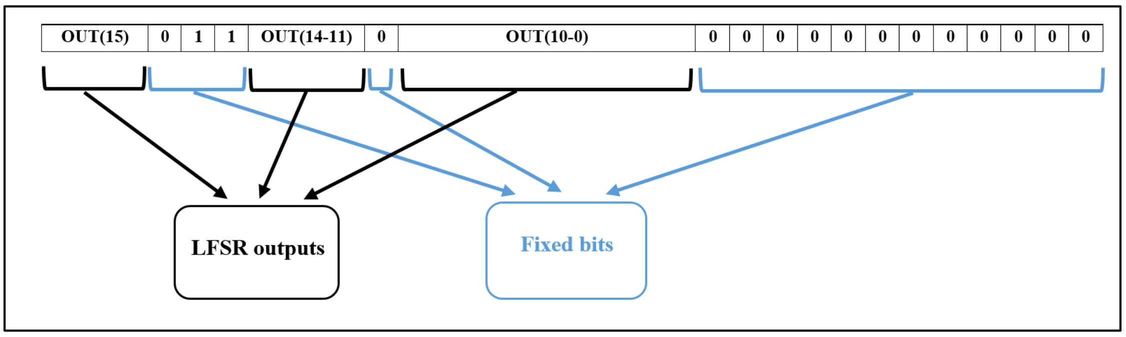


Figure 3-9: The32-bit format of the randomized weight values.

Since our ANN requires 1072 weights to store, we will need to iterate 1072 times through the same procedure described above to complete the initialization process of the weights by incrementing the address value after checking if it did not reach the last one (1071) and repeat the steps of loading the SRAM address with the appropriate value and store its corresponding SRAM input. Once completed, the ann_mode is assigned the idle state.

2) Learning Phase

When the learning phase begins, the ann_mode, which was previously in the idle state, is set to learn, and the learning flag is given the HIGH value to indicate that it has started. It sequentially executes many substates to perform the following computations:

1. Calculation of the weighted sum and the output from the sigmoid function for every perceptron in the hidden layer.
2. Calculation of the weighted sum and the output from the sigmoid function for all perceptrons in the output layer.
3. Subtraction output from its corresponding training target for each perceptron in the output layer to obtain the error estimation.
4. Computation of Mean Squared Error (MSE) by adding up the errors calculated for all the perceptrons in the output layer.
5. For each perceptron in the output layer, the derivative of the sigmoid function is calculated and multiplied by the error.
6. To update the weights, calculation of delta multiplied with weight for each of the input connections for all the perceptrons in the output layer should be performed to obtain new weights based on the delta multiplied with weight for these connections.
7. Utilizing the calculated deltas for the perceptrons in the output layer, it determines the error for each hidden layer perceptron.
8. For each perceptron in the hidden layer, it determines delta as the derivative of the sigmoid function multiplied by the error.
9. Updates the weight based on the delta weight for these connections and calculates the delta weight for each of the input connections for each of the perceptrons in the hidden layer.

3) Running Phase

At this stage, the ANN is already trained and reached the optimal values for the weights through the backpropagation process and is fully capable of accomplishing the pattern recognition task, therefore, the ready signal is set to HIGH and LED[8] lights up to show to the user that it is ready and can be tested. When the user presses KEY [0], the ANN mode is set to run and starts the recognition, by executing a number of operations in order to perform the following:

1. Calculation of the weighted sum and sigmoid function's output for every perceptron in the hidden layer.

2. Calculation of the weighted sum and the output from the sigmoid function for all perceptrons in the output layer.

4) Idle phase

Whenever the ANN completes the execution of all instructions in each of the three previously mentioned phases, it enters an idle state in which no computations are undertaken and no hardware is being used. It aims to reduce energy consumption.

2.3 16-Bit Linear Feedback Shift Register

The linear feedback shift register is a pseudo-random numbers generator (PRNG), also known as deterministic random bit generator (DRBG), is an algorithm that produces a sequence of numbers having approximately the same properties as of random numbers sequences, a type of shift registers that constitute of a series of flip-flops connected in a sequential architecture. The generation process of these numbers is not completely random since it is based on an initial inserted value, which is called the seed. The first output is calculated using the seed value processed with a linear function that is usually composed of some XOR gates, and the subsequent outputs are dependent on their previous value that is fed back to the system as an input. Since, the register has a finite number of possible states; it will eventually enter a repeating cycle. Although, an LFSR with a well-chosen number of bits and an appropriate linear function can produce a pseudo-random sequence with a long cycle. For an n-bit LFSR, the maximum Period is 2^n-1. There exist two types of LFSR implementation, which are the Fibonacci and Galois implementations. The main difference between them is the arrangement of gates (usually XOR gates) in the circuit. In the former, the XOR gates are cascaded, resulting in a bigger propagation delay that affects the timing performance of the circuit due to its architecture. While in the latter, the XOR gates are placed between two consecutive registers, allowing for parallel computations and a minimum propagation delay, equivalent to that of a single XOR gate.

We have chosen the 16-bit Galois LFSR version (**Figure 3-10**) and utilized it primarily in our design to initialize the artificial neural network's weights with random values. It contains only two input ports and one output port, which are the 50MHz clock for synchronization, the reset signal that is assigned the value of KEY [0], and the 16-bit output port, respectively.**Figure3-11**shows its circuit diagram.

In the implementation, we used a seed value of "**ACE1**" in hexadecimal notation which is equivalent to "1010 1100 1110 0001"in binary notation. When the reset signal is HIGH, we assign the seed value as an initial value to start the pseudo random generation of the LFSR, in order to have a maximum period (65,535), we defined our linear function to be the XORing of the following pair of bits:

1- (12, 1) as the input to bit number 11.
2- (14, 1) as the input to bit number 13.
3- (15, 1) as the input to bit number 14.

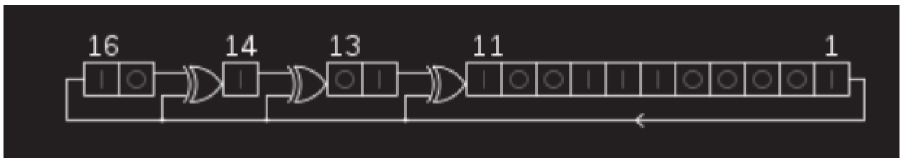


Figure 3-10: The 16-bit GALOIS LFSR diagram.

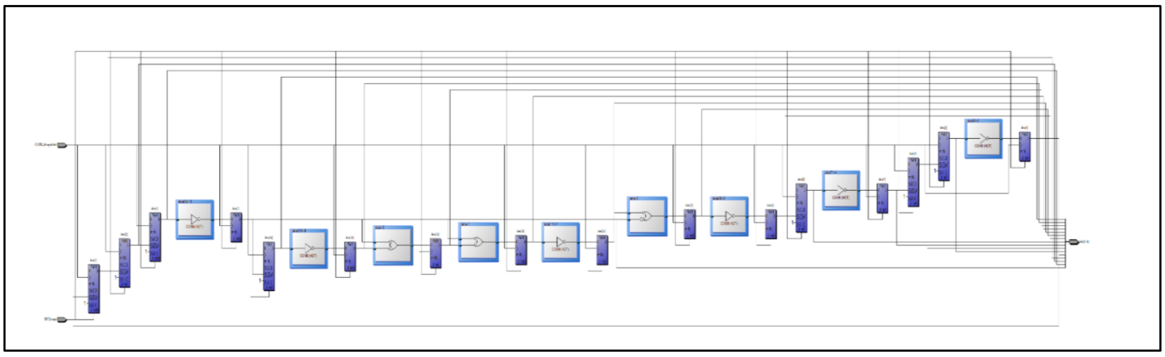


Figure 3-11: The circuit diagram of the LFSR component in Quartus software.

2.4 Float ALU Component

The floating-point arithmetic logic unit is a synthesized block built using the IP cores since the Altera DE2-115 FPGA board does not have an onboard built-in floating point ALU. It includes five subcomponents that are a floating-point adder, subtractor, multiplier, divider, and a floating-point exponentiator. It is the major element that handles all of the operations required for the ANN to operate properly. Its IN/OUT structure consists of three input ports that are:

- Reset that is given the value of KEY [0].
- Clock signal which is the available 50 MHz oscillator on the board.
- The ALU mode that determines which operation to execute (addition, subtraction, multiplication…etc).

And two output ports that are the ready signal and the 32-bit floating point result of the operations. All of its subcomponents have an additional input port that is the clock enable signal that activates them when being selected. **Figure 3-12** illustrates the IN/OUT structure.

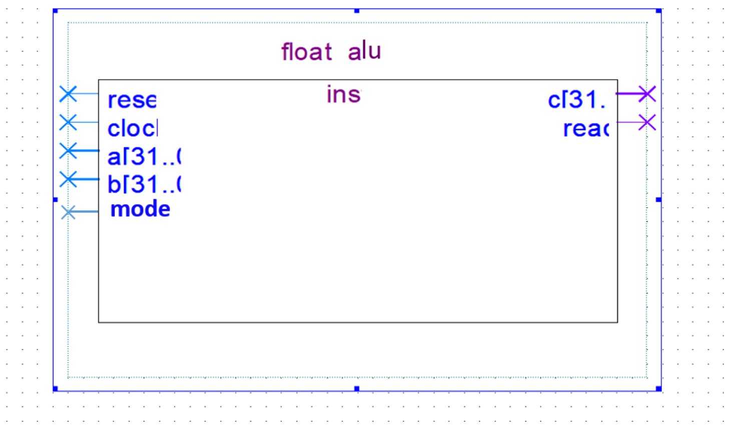


Figure 3-12: Block diagram representation of the ALU.

When we programmed this ALU in VHDL, we utilized the subcomponents in the structural modelling style, meaning that we declared them as components and mapped their ports with the corresponding signals. Likewise, we declared the ALU as a component and implemented an FSM that represents how it operates. It is comprised of six states that refer to the ALU modes, which are selected one at a time by the ANN based on the computations needed. The six states are:

-Idle: The ALU does not execute any operation and remains unused by the ANN.

-Add: The ALTFP_ADD_SUB IP core is employed to perform the floating-point addition.

-Sub: The ALU uses the ALTFP_ADD_SUB IP core to do the floating-point subtraction.

-Mul: The multiplication operation is executed with the help of the ALTFP_MULT IP core.

-Div: The division operation is undertaken by the ALU using the ALTFP_DIV IP core.

-Exp: The exponentiation operation is carried out by utilizing the ALTFP EXP IP core.

All of the ALU subcomponents require different clock cycle delays to work properly; therefore, we designed the ANN to wait for a flexible number of clock cycles to obtain the ALU results before employing it for further processing.

2.5 SRAM Component

The Static Random Access Memory or SRAM is a known type of semiconductor random access memory that employs an array of latching circuitries (flip-flops) to store each bit. It retains storage bits as long as power is being provided. Unlike its counterpart Dynamic Random Access Memory or DRAM, which requires continuous refreshment, the SRAM does not have this necessity, resulting in low power consumption and higher performance. Consequently, we used the on-board SRAM chip on the DE2-115 board in our project to store the 1072 32-bit floating-point weight values. The DE2-115 board has 2MB SRAM memory with 16-bit data width. Being featured with a maximum performance frequency of about 125MHz under the condition of standard 3.3V single power supply makes it suitable of dealing with high-speed media processing applications that need ultra-data throughput. The related schematic is shown in **Figure 3-13** [20]. In its VHDL code, we added the asynchronous reset signal (KEY [0]) and the 50MHz clock signal input ports.

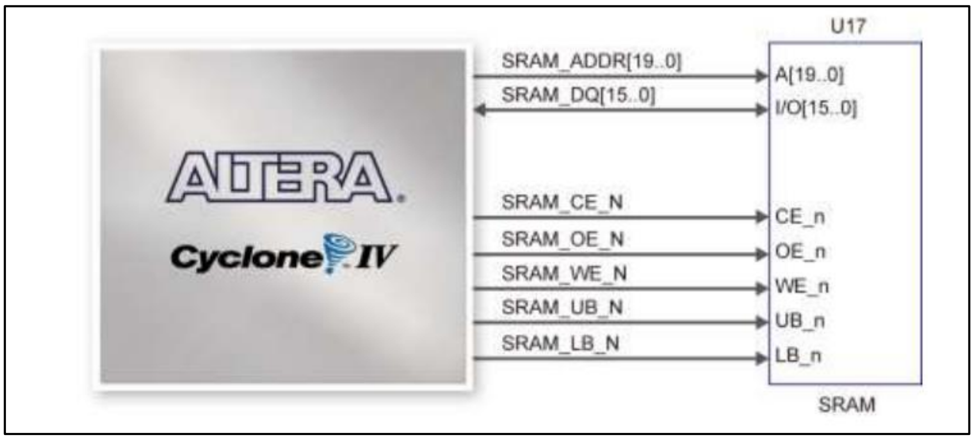


Figure 3-13: Connections between FPGA and SRAM [20].

Although, since we needed a total of 4,288 bytes to store all the weights, we only employed one fourth of the available SRAM chip memory to have an efficient program and save on hardware utilization. Therefore, we set the address bus port to be 18bits wide. It operates in three different modes: idle, write, and read that are selected by the ANN depending on the current running operations. The writing to and reading from the SRAM are managed by an FSM that is designed with six states (**Figure 3-14**), which are:

-Init: The SRAM input (SRAM_DQ), ready signal and write enable are all initialized to zero in this state.

-Idle: The SRAM module remains inactive and unused by the ANN, meaning that no writing or reading operation execution.

-Read_low: When the mode is set to read by the ANN, the SRAM masks the two most significant bytes of the 32-bit weight value and places on its output port the two least significant bytes to send them to the ANN unit. Once finished, the next state is set to read_high.

-Read_high: The SRAM masks the two least significant bytes, and places on its output port the value of the two most significant bytes of the weight value to complete the reading operation. Then, the state goes back to idle.

-Write_low: When the mode is set to write, the SRAM receives inputs from the ANN block and places the two least significant bytes in its input port by masking the two most significant bytes in order to store 16-bit from the weight value at a time. Following that, the next state is assigned to write_high state.

-Write_high: In this state, the SRAM stores the two most significant bytes of the weight's value by masking the least significant ones. When the operation is completed, the state is sent back to idle.

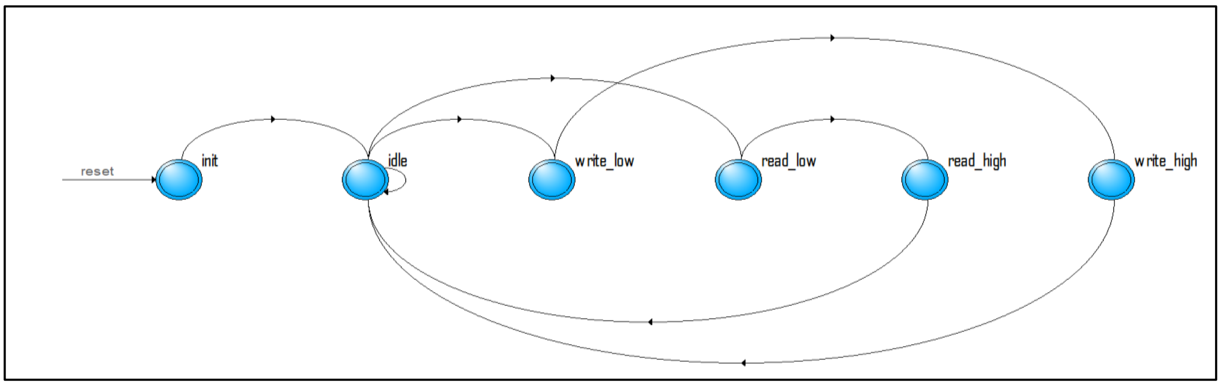


Figure 3-14: SRAM Finite State Machine diagram representation.

2.6 LCD Component:

The LCD unit with dimensions of 80.0×36.0×13.5(MAX) mm and a maximum of 16 characters×2 Lines as the number of characters, has built-in fonts and can be used to display text by sending appropriate commands to the display controller called HD44780 [20]. The controller has two 8-bit registers, an instruction register (IR) and a data register (DR) that manage the operation of the LCD. The schematic diagram of the LCD module showing connections to the Cyclone IV E FPGA is given in **Figure 3-15** [20].We mainly employed this component in our project to enter the letters on the 4×4 grids using the toggle switches and to show the output of the recognition process in each of the two grids. In the VHDL code, we utilized an FSM to handle the reading and writing of our data to the LCD.

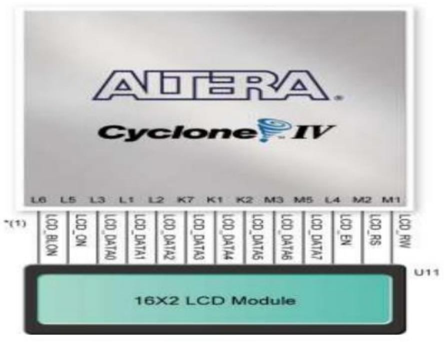


Figure 3-15: Connections between the LCD module and Cyclone IV E FPGA [20].

2.7 Display Control Component

The display element is a control unit for the LCD; it has five input ports and two output ports that are respectively: the asynchronous reset KEY [0] input, the 50MHz clock signal, the display mode input port, the 18 toggle switches inputs, the output class from the pattern recognition process as an input, and the two output ports; character graphics (**lcd_cg**) and pattern switches display (**lcd_dd**). Its block diagram representation is illustrated on **Figure 3-16.** In its VHDL code, we defined the character graphics, which are the customized pixels since we represented the letters on a 4×4 grid, meaning that we specified an area of **two** rows and **four** columns in the LCD for one grid (**Figure 3-17**). In addition to that, we assigned each switch to its corresponding pixel and based on the state of the switches “0” and “1”, one grid is selected at a time to be populated by the remaining 16 switches [17 - 2]. It has three display modes that depend on the state of the ANN. When it is training, the display mode is also set to training, and the message "train" is written on the rightmost portion of the LCD. When the KEY [3] is pressed, the display mode is set to running, and the message "run" is displayed on the LCD. While in the idle state, it displays the output class.

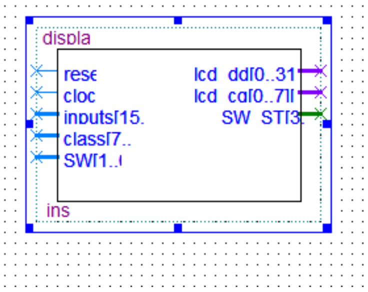


Figure 3-16: Block diagram of display control unit.

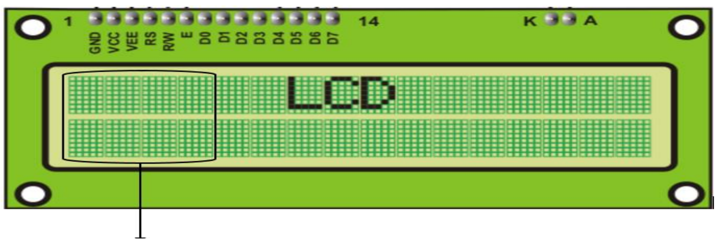


Figure 3-17: LCD grid dimensions.

2.8 Motor Circuit

The motor circuit is a designed off-chip system that serves as a demonstration of the proper functioning of the pattern recognizer system. It consists of a ULN2003AN motor driver hardware and a 28BYJ-48 unipolar stepper motor (**Figure2-20**) that requires a 5v power supply, which is suitable for our application since it can be provided by the DE2-115 board. The driver PCB is connected to the board through the GPIO pins that allow for a communication between the FPGA and the motor. The interfacing circuit is shown on **Figure 3-18.**

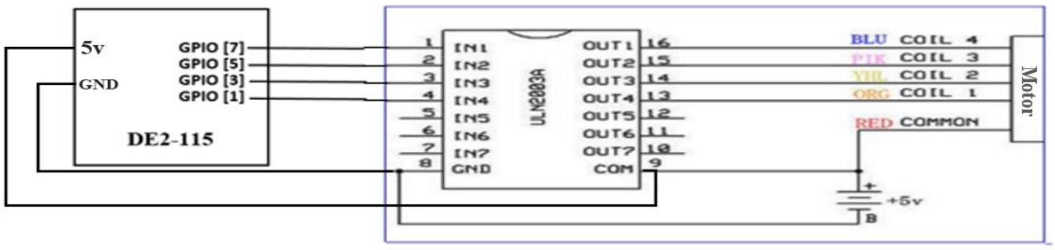


Figure 3-18: The interfacing of the Motor circuit.

When the pattern **"ON"** is detected on the two 4×4 LCD grids the motor is turned ON in full step mode and when the **"OF**" pattern is entered, the motor is turned OFF. This behaviour is controlled by the PR block that is responsible for sending and setting the value of the motor's enable signal based on the detected letters on the grids. In case the enable signal is set to '1', a 3-bit counter starts counting using the 50MHz clock and in each state, the GPIO pins connected to ULN2003AN [IN1-IN4] ports are given a certain value to energize the appropriate coils which are shown in **Table 3-1**. However, if the enable signal is zero, the counter is disabled and stays at state "000" and the motor is off.

Counter state	GPIO(7)	GPIO(5)	GPIO(3)	GPIO(1)
000	1	0	0	0
001	1	1	0	0
010	0	1	0	0
011	0	1	1	0
100	0	0	1	0
101	0	0	1	1
110	0	0	0	1
111	1	0	0	1

Table 3-1: GPIO pins values.

3 Altera Floating Point Megafunctions

As design complexities increase, the use of vendor-specific intellectual property (IP) blocks has become a common design methodology. Altera offers parameterizable and specifically optimized floating-point megafunctions that comply with the IEEE-754 standard for its device architectures. Using megafunctions instead of coding our own logic saves valuable design time. Altera's functions enable efficient and faster logic synthesis and device implementation. Its General Features are:

- Support for floating-point formats: single precision, double precision etc.

- Input support for not-a-number (NaN), infinity, zero, and normal numbers.

- Support for round-to-nearest-even rounding mode.

- Optional asynchronous input ports including asynchronous clear (aclr) and clock enable (clk_en).

- Denormal number inputs are not supported by Altera's floating-point megafunctions. When given a denormal value as an input, the megafunction forces the value to zero and handles it as such before performing any operation.

3.1 ALTFP_COMPARE

The Altera floating-point compare megafunction implements comparison functions, it offers many features such as seven-status output ports where each one denotes the result obtained for a certain comparison operation. Its block diagram is depicted on **Figure 3-19**. In this project, the single precision, the agb_output port and the optional input ports: asynchronous clear "**aclr"** ", and a clock enable "**clk_en**" , which, as its name implies, enables the comparison operation to occur when the port is asserted high are used.

Let us suppose two floating-point numbers A and B that are represented by equations (3.1) and (3.2).

A = (–1) Sa × 2Ea × 1.Ma…………………………..**Eq. (3.1)**
B = (–1) Sb × 2Eb × 1.Mb…………………………..**Eq. (3.2)**

These equations have the following values:

- Sa and Sb are sign bits
- Ea and Eb are exponent values
- Ma and Mb are mantissa bits

The output of the floating-point comparator is obtained from the result of comparing input A and input B using equation 3.3.

Agb = (–1) Sa × 2Ea × 1.Ma > (–1) Sb × 2Eb × 1.Mb………………………..**Eq. (3.3)**

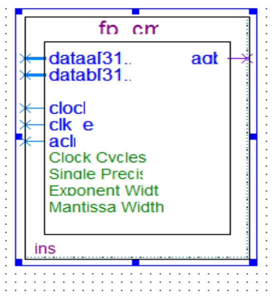
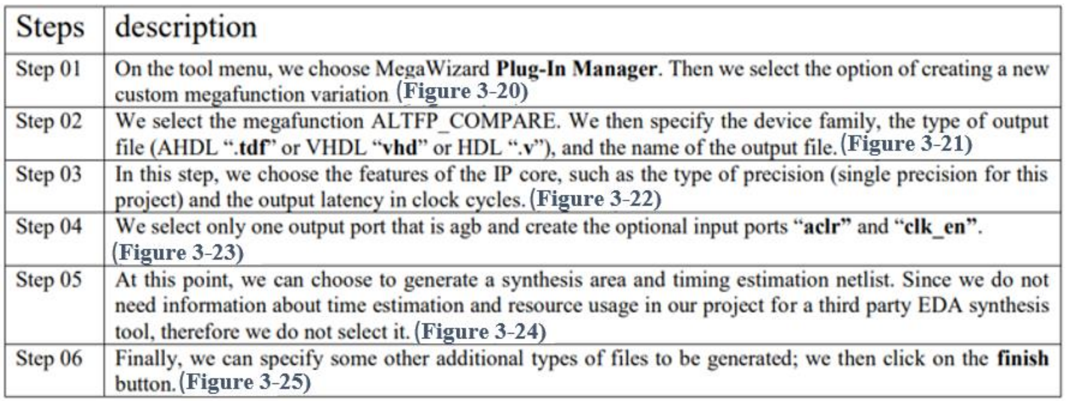


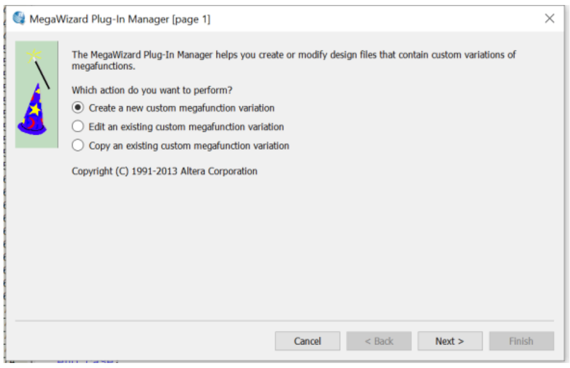
Figure 3-19: The ALTFP_COMPARE IP core block diagram.

In order to include this IP core in the design, the corresponding steps to follow are summarized in **Table3-2.**



Steps	description
Step 01	On the tool menu, we choose MegaWizard **Plug-In Manager**. Then we select the option of creating a new custom megafunction variation (**Figure** 3-20)
Step 02	We select the megafunction ALTFP_COMPARE. We then specify the device family, the type of output file (AHDL "**.tdf**" or VHDL "**vhd**" or HDL ".v"), and the name of the output file. (**Figure** 3-21)
Step 03	In this step, we choose the features of the IP core, such as the type of precision (single precision for this project) and the output latency in clock cycles. (**Figure** 3-22)
Step 04	We select only one output port that is agb and create the optional input ports "**aclr**" and "**clk_en**". (**Figure** 3-23)
Step 05	At this point, we can choose to generate a synthesis area and timing estimation netlist. Since we do not need information about time estimation and resource usage in our project for a third party EDA synthesis tool, therefore we do not select it. (**Figure** 3-24)
Step 06	Finally, we can specify some other additional types of files to be generated; we then click on the **finish** button. (**Figure** 3-25)

Table3-2: The procedure of including the ALTFP_COMPARE IP core.



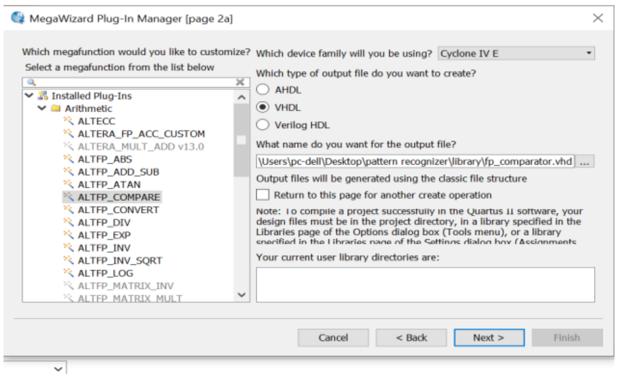
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

Which action do you want to perform?

- [✓ Create a new custom megafunction variation]
- Edit an existing custom megafunction variation
- Copy an existing custom megafunction variation

Copyright (C) 1991-2013 Altera Corporation

Figure 3-20: MegaWizard Plug-In Manager window.



Which megafunction would you like to customize?
Select a megafunction from the list below

Which device family will you be using? Cyclone IV E

Which type of output file do you want to create?

- AHDL
- [✓ VHDL]
- Verilog HDL

What name do you want for the output file? \Users\pc-dell\Desktop\pattern recognizer\library\fp_comparator.vhd

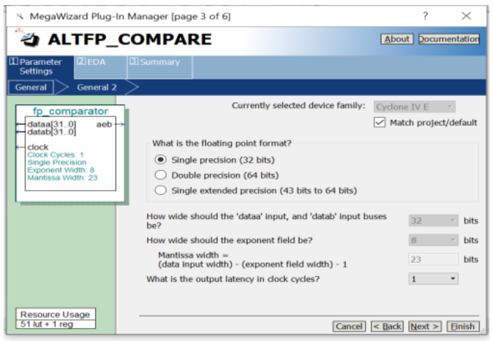
Output files will be generated using the classic file structure

Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of the Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments

Your current user library directories are:

Figure 3-21: IP core selection window.



Currently selected device family: Cyclone IV E

Match project/default ✓

What is the floating point format?

- [✓ Single precision (32 bits)]
- Double precision (64 bits)
- Single extended precision (43 bits to 64 bits)

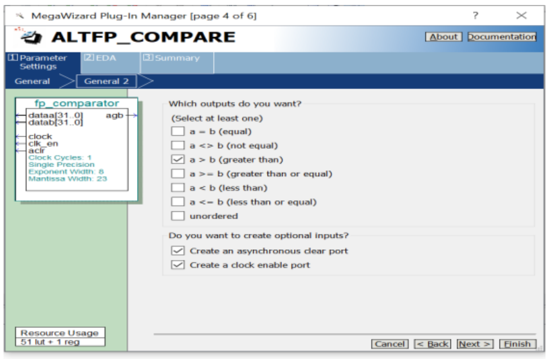
How wide should the 'dataa' input, and 'datab' input buses be? 32 bits

How wide should the exponent field be? 8 bits

Mantissa width = (data input width) - (exponent field width) - 1 23 bits

What is the output latency in clock cycles? 1

Figure 3-22: Features selection Window.



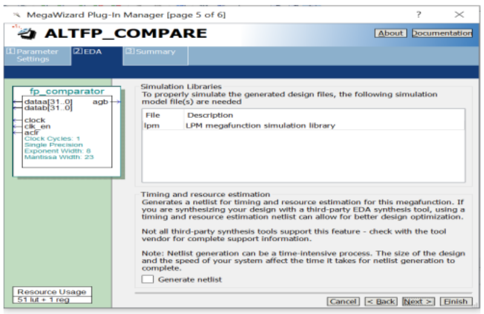
Which outputs do you want?
(Select at least one)

- a = b (equal)
- a <> b (not equal)
- [✓ a > b (greater than)]
- a >= b (greater than or equal)
- a < b (less than)
- a <= b (less than or equal)
- unordered

Do you want to create optional inputs?

- [✓ Create an asynchronous clear port]
- [✓ Create a clock enable port]

Figure 3-23: Output ports selection window.



Simulation Libraries
To properly simulate the generated design files, the following simulation model file(s) are needed

File	Description
lpm	LPM megafunction simulation library

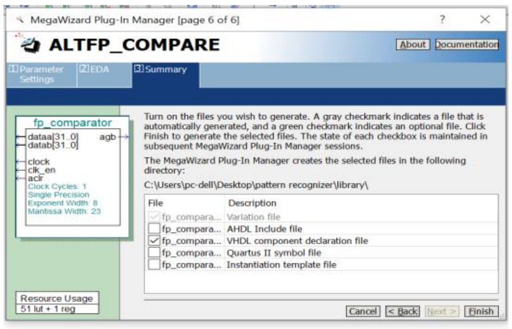
Timing and resource estimation
Generates a netlist for timing and resource estimation for this megafunction. If you are synthesizing your design with a third-party EDA synthesis tool, using a timing and resource estimation netlist can allow for better design optimization.

Not all third-party synthesis tools support this feature - check with the tool vendor for complete support information.

Note: Netlist generation can be a time-intensive process. The size of the design and the speed of your system affect the time it takes for netlist generation to complete.

Generate netlist

Figure 3-24: Synthesis area and timing estimation netlist option selection window.



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:

C:\Users\pc-dell\Desktop\pattern recognizer\library\

File	Description
✓ fp_compara...	Variation file
fp_compara...	AHDL Include file
✓ fp_compara...	VHDL component declaration file
fp_compara...	Quartus II symbol file
fp_compara...	Instantiation template file

Figure 3-25: Additional files generation window.

3.2 ALTFP_EXP

The Floating Point Exponent (ALTFP_EXP) megafunction calculates the exponential value of a given input. The representation of this IP core is illustrated on **Figure 3-26.** It is the main building block for the implementation of the sigmoid activation function.

The same procedures outlined in **Table 3-2** are used to add this IP core to our architecture. The megafunction provides many other features as the previously mentioned ones. However, we will only use the single precision format and the optional input ports, asynchronous clear ("aclr"), and a clock enable ("clk en").

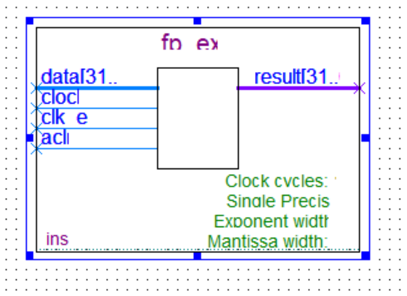


Figure 3-26: The block diagram representation of the ALTFP_EXP IP core.

3.3 ALTFP_ADD_SUB

The ALTFP_ADD_SUB implements a floating-point adder/substractor. Given the two floating-point numbers A and B in equations 3.1 and 3.2, the output "**out**" is obtained from the sum or difference as expressed by equation 3.4:

Out= (–1) Sa × 2Ea × 1.Ma ± (–1) Sb × 2Eb × 1.Mb……………………………**Eq. (3.4)**

Only when both exponents of two floating-point integers are equal can the mantissa of those numbers be added or subtracted. Therefore, it is implemented using the following procedures:

- Input verification and alignment:

The inputs should be verified first to see whether they are denormal numbers; if so, the inputs should be forced to zero, the output is a **don't care** and ignored and the indefinite output flag is set. In case they are not denormal numbers the followed steps are:

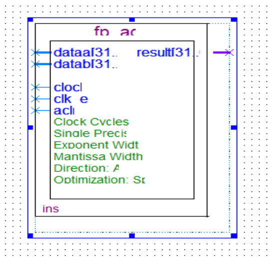
- Finding the input with the smallest exponent.
- Obtain the difference between both exponents.
- Expand by 1-bit the mantissa field of the inputs since they are normalized numbers.
- Right shift the number with the smallest exponent by an amount equal to the difference of the two exponents using a barrel shifter.
- At this stage, the two numbers should have the same exponent, which is equal to the one of the larger number.
- The sign bit of the inputs remain unchanged.

- Addition or subtraction of the expanded mantissa of the two numbers:

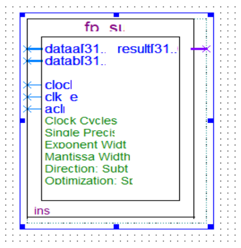
The two inputs with expanded mantissa fields are added or subtracted based on the status of the add_sub port and the sign bit field of both numbers.

- Renormalizing the result by shifting left and decrementing the exponent.
- Rounding the result to the nearest even.
- Checking for exceptions and set output flags accordingly.

When the sum or difference of the inputs produces a denormal number, the underflow and denormal flags are set. The block diagram representation of the instantiated adder and substractor modules are shown in **Figure 3-27**.



(A): Adder.



(B): Subtractor.

Figure 3-27: The block diagram of IEEE-754 floating-point adder/subtractor IP cores.

We instantiated this IP core twice in our design, once as an adder only and later as a substractor and we did not utilize any exception handling ports.

3.4 ALTFP_MULT

This megafunction implements floating-point multiplier functions. When it takes A and B of equations (3.1) and (3.2) respectively as inputs it generates the result "R" of the floating-point multiplication based on equation 3.5:

R= (Ma × 2Ea) × (Mb × 2Eb) = (Ma × Mb) × 2Ea+Eb…………………**Eq. (3.5)**

Where:
- Ea, Eb are the exponent bits of A and B respectively.
- Ma, Mb are the mantissa bits of A and B subsequently.

The result of the multiplication has a sign bit equal to the exclusive OR of the inputs' sign bits, a mantissa equal to the multiplication of the inputs' mantissa fields, and an exponent field that is the sum of the inputs' exponents. However, when the exponents of the inputs are added, an extra bias occurs and it should be removed to have the correct value of the result's exponent. The following calculations demonstrate how their addition causes an extra bias:

Let us express the inputs' exponents as:
-Exp_A = Exp_A_actual + bias
-Exp_B = Exp_B_actual+ bias, where the bias for single precision is 127

Their addition "ad" is:

Ad = Exp_A + Exp_B = Exp_A_actual + Exp_B_actual + bias + bias

Ad = Exp_A_actual + Exp_B_actual+ 2×bias.

The term 2×bias should be reduced to 1×bias only to remove the excess and get the correct result.

when we instantiated this megafunction in our design, we did not use any exception signal and as in the previous ones, we selected the optional ports **aclr** and **clk_en**. **Figure 3-28** shows the block diagram representation of this IP core.

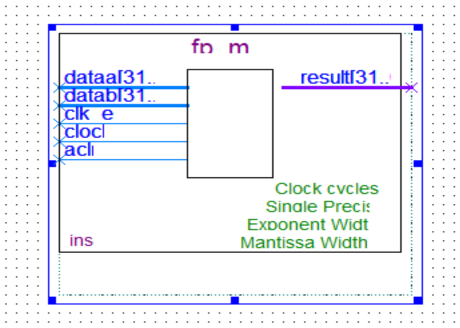


Figure 3-28: Block diagram of ALTFP_MULT IP core.

3.5 ALTFP_DIV

This megafunction implements a floating-point division function. When given inputs A and B, it calculates the result "R" based on equation 3.6:

$$R = \frac{(-1)\ Sa \times 2Ea \times 1.Ma}{(-1)\ Sb \times 2Eb \times 1.Mb} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{Eq. (3.6)}$$

Where:

-Sign of R = Sign bit of A ⊕ Sign bit of B
-Exponent of R = Exponent of A – Exponent of B + Bias

$$\text{- Mantissa} = \frac{\text{Mantissa of A}}{\text{Mantissa of B}}$$

The bias is added to the exponent because when the subtraction operation between the input exponents is performed, the obtained result will not contain any bias, so it is reintroduced to have a correct exponent for the result.

In the instantiation of this IP core,we have only added the optional ports “**aclr**” and “**clk_en**” as in the previous megafunctions. **Figure 3-29 shows** its block representation.

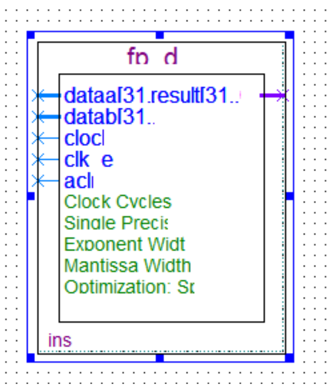


Figure 3-29: Block diagram of ALTFP_DIV IP core.

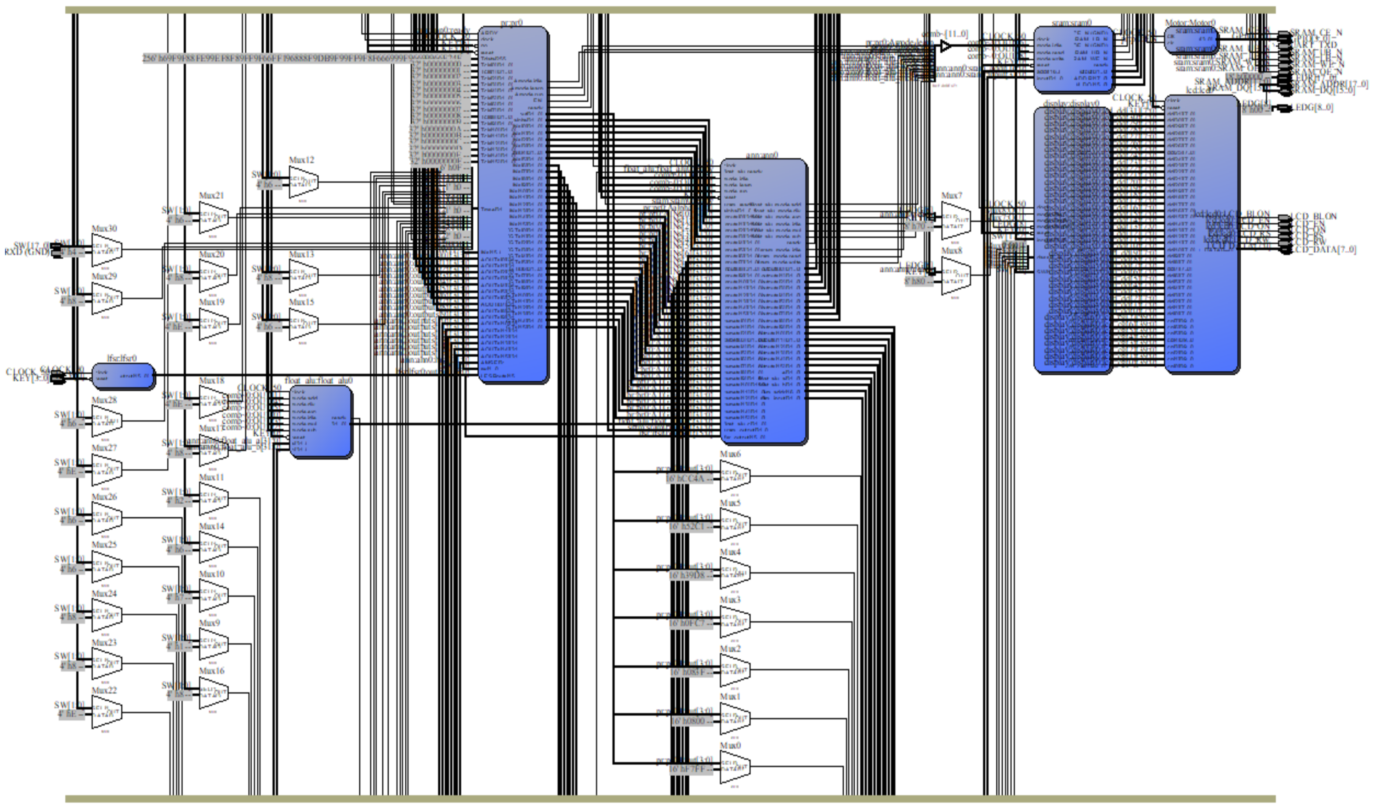
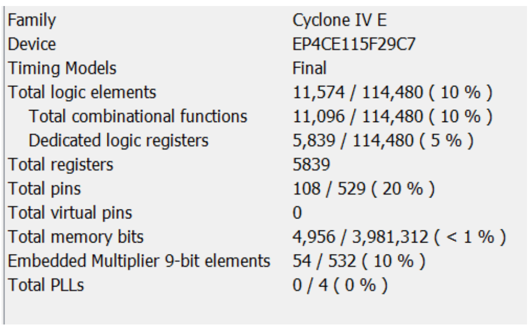


Figure 3-30: The synthesized system on FPGA shown in the RTL viewer of Quartus II software.

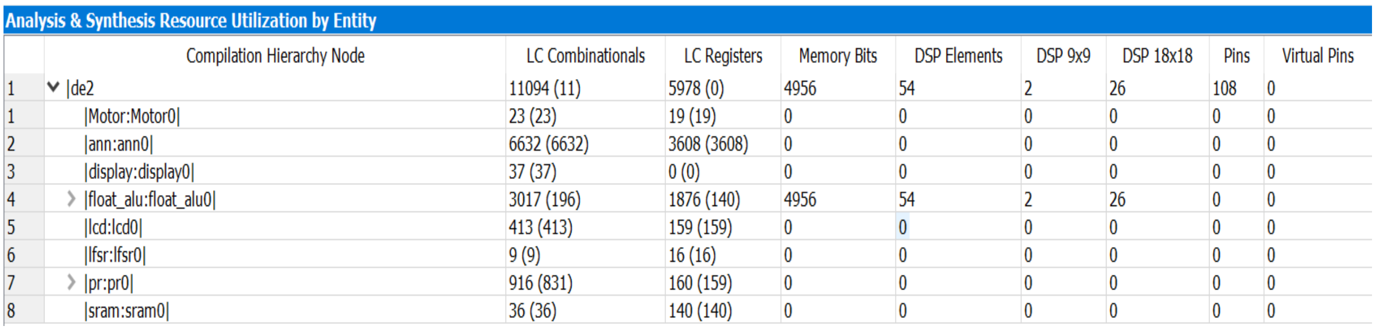
4 Resource Usage

At the end of the compilation process of all the project VHDL files, Quartus II software generates some reports that summarize the resource usage on the EP4CE115F29C7Cyclone IV E FPGA device which has a total of 114 480 logic elements. It utilized 10% of the total LEs; in other words 11 574 of LEs and the synthesized system displayed in the RTL viewer is shown on **Figure 3-30**. The resource usage report for our project is presented on **Figure 3-31**.



Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	11,574 / 114,480 (10 %)
Total combinational functions	11,096 / 114,480 (10 %)
Dedicated logic registers	5,839 / 114,480 (5 %)
Total registers	5839
Total pins	108 / 529 (20 %)
Total virtual pins	0
Total memory bits	4,956 / 3,981,312 (< 1 %)
Embedded Multiplier 9-bit elements	54 / 532 (10 %)
Total PLLs	0 / 4 (0 %)

(A): The flow summary.



Analysis & Synthesis Resource Utilization by Entity

	Compilation Hierarchy Node	LC Combinationals	LC Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins
1	|de2	11094 (11)	5978 (0)	4956	54	2	26	108	0
1	|Motor:Motor0|	23 (23)	19 (19)	0	0	0	0	0	0
2	|ann:ann0|	6632 (6632)	3608 (3608)	0	0	0	0	0	0
3	|display:display0|	37 (37)	0 (0)	0	0	0	0	0	0
4	|float_alu:float_alu0|	3017 (196)	1876 (140)	4956	54	2	26	0	0
5	|lcd:lcd0|	413 (413)	159 (159)	0	0	0	0	0	0
6	|lfsr:lfsr0|	9 (9)	16 (16)	0	0	0	0	0	0
7	|pr:pr0|	916 (831)	160 (159)	0	0	0	0	0	0
8	|sram:sram0|	36 (36)	140 (140)	0	0	0	0	0	0

(B): Resource usage by entity.

Figure 3-31: Resource usage from Quartus II software.

5 Experimental Results

One of our main research design concerns was to achieve a fully hardware implementation of an artificial neural network on FPGA that is trained with a dataset defined inside a VHDL file and tested by the user with an interactive interface. Therefore, this project involved the employment of many features provided by the DE2-115 board such as the SRAM module, the LCD display, the 18 toggle switches etc.

After loading the program to the board, we observed satisfactory system performance that met our expectations, revealing that the design procedure was successful. At the start, the LCD displays the “train” message, which notifies the user that the ANN block is still in the learning phase, as indicated by **Figure 3-32**. Then, the system becomes ready to receive user inputs from the switches to enter the desired pattern on the selected grid and once KEY [3] is pressed, the recognition phase begins, and after few seconds, around approximately 10secs, the output class is displayed on the rightmost portion of the LCD as shown on **Figure 3-33**. Whenever, the pattern ON is detected on the two LCD grids, an enable signal with a value of ‘1’ is sent to the motor control circuit, which in turn sets the motor in full step drive. While, if the pattern is OF, the motor control circuit is deactivated by an enable signal of value ‘0’ and keeps the motor inactive by preventing the occurrence of the right coils energizing sequence that yields to its movement. **Figures 3-33 (A)** and **(B)** demonstrates the system’s output for the two mentioned patterns (OF and ON respectively).

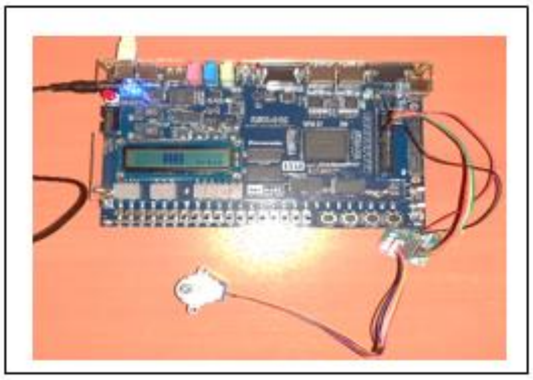
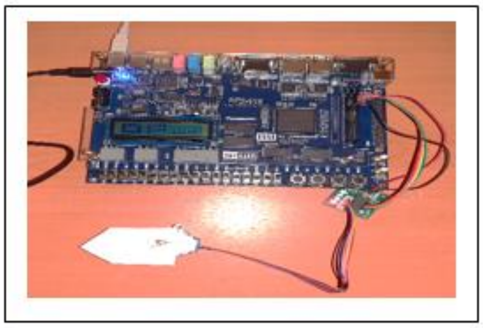
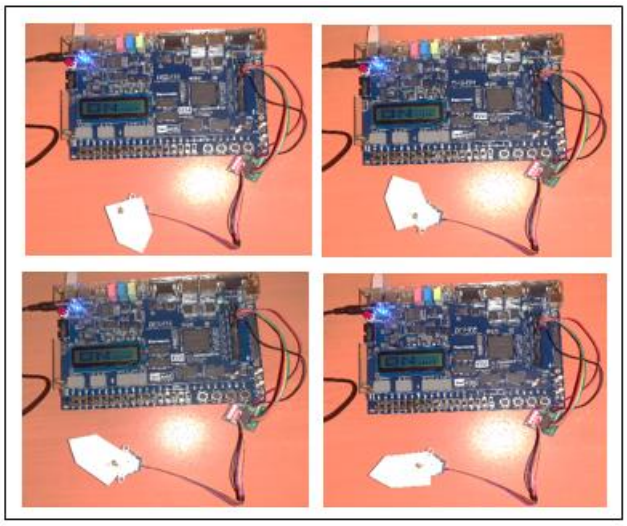


Figure 3-32: The real system shot in the training phase.



(A): “OF” pattern system’s response



(B): “ON” pattern system’s response.

Figure 3-33: Real system output.

Conclusion

This report presents the design and implementation of an artificial neural network on a reprogrammable platform, the EP4CE115F29C7Cyclone IV E FPGA. The IEEE-754 single precision floating-point format is utilized for the representation of data. The 16 mentioned characters can be represented on the 4x4 grids of the LCD. The weights are randomly initialized by an implemented 16-bit Galois LFSR with a maximum period of 65,535 and then stored in the built-in onboard SRAM.
The pattern recognition Machine learning application is implemented to validate the capability of the ANN to produce satisfying results with the help of a control unit labelled as PR block that handles its working. Moreover, an illustrative circuit that consists of a stepper motor and its driver are added to the system to serve as a physical demonstration. No work is ever complete; there is always room for possible improvements to be made in the future. Some useful additions to enhance the performance of the system are:

- Increasing the dimensions of the grids, by employing an external display to have more freedom in the representation of patterns and include more than 16 letters.
- An interface can be implemented to enable the ANN to recognize hand-written characters.
- A text-to-speech application can be designed to transform the recognized characters into audio.

Ultimately, this project lays a solid foundation for future work in the field of embedded pattern recognizer circuits. It tackled the most difficult parts of such systems, essentially, a scalable ANN architecture with all the necessary computational power that learns and outputs results within few seconds.

References:

[1] John, Soldatos. “The Embedded Machine Learning Revolution: The Basics You Need to Know.” Wevolver, www.wevolver.com/article/the-embedded-machine-learning-revolution-the-basics-you-need-to-know (Accessed 4 Aug. 2022).

[2] Lacey, Griffin, et al. “Deep Learning on FPGAs: Past, Present, and Future.” ArXiv, vol. v1, 13 Feb. 2016, pp. 4–5, 1602.04283.

[3] Bacon, David F., et al. “FPGA Programming for the Masses.” Communications of the ACM, vol. 56, no. 4, Apr. 2013, pp. 56–63, 10.1145/2436256.2436271.

[4] Rosenfeld, Azriel, and Harry Wechsler. “Pattern Recognition: Historical Perspective and Future Directions.” International Journal of Imaging Systems and Technology, vol. 11, no. 2, 2000, pp. 1–2, 3.0.co;2-j">10.1002/1098-1098(2000)11:2<101::aid-ima1>3.0.co;2-j.

[5] “Types of Artificial Intelligence - Javatpoint.” Www.javatpoint.com, 2011, www.javatpoint.com/types-of-artificial-intelligence (Accessed 10 Aug. 2022).

[6] R. Saracco, "Computers keep getting better … than us," IEEE Future Directions, 2018.

[7] Ziyad, Mohammed. Artificial Intelligence Definition, Ethics and Standards. 2019.

[8] Mitchell, Tom. “Machine Learning.” New York: McGraw Hill, 1997.

[9] Raj, Ravish. “Supervised, Unsupervised, and Semi-Supervised Learning with Real-Life Usecase.” Www.enjoyalgorithms.com, www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning (Accessed 15 Aug. 2022).

[10] Aggarwal, Charu C. Neural Networks and Deep Learning a Textbook. Cham, Springer International Publishing, 2018.

[11] Kinsley, Harrison, and Daniel Kukieła. Neural Networks from Scratch in Python. 2020, p. 9.

[12] “Hidden Layer.” DeepAI, 17 May 2019, deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning (Accessed 9 Aug. 2022).

[13] Artificial Neural Networks • Threshold Units • Gradient Descent • Multilayer Networks • Backpropagation • Hidden Layer Representations • Example: Face Recognition Advanced Topics CS 8751 ML & KDD Artificial Neural Networks 2 Connectionist Models Consider Humans.

[14] Hassim, Yana MazwinMohmad, and RozaidaGhazali. "Training a Functional Link Neural Network Using an Artificial Bee Colony for Solving a Classification Problems." ArXiv:1212.6922 [Cs], 31 Dec. 2012, arxiv.org/abs/1212.6922. Accessed 19 July 2022.

[15] "What Is FPGA? FPGA Basics, Applications and Uses | Arrow.com." Arrow.com, 24 Sept. 2018, www.arrow.com/en/research-and-events/articles/fpga-basics-architecture-applications-and-uses. Accessed 10 July 2022.

[16] Akhtar. "FPGA Architecture." Invent Logics, 16 Apr. 2014, allaboutfpga.com/fpga-architecture/ (Accessed 22 July 2022).

[17] "Know about FPGA Architecture and Their Applications." ElProCus - Electronic Projects for Engineering Students, 18 Sept. 2014, www.elprocus.com/fpga-architecture-and-applications/ (Accessed 25 July 2022).

[18] "FPGA Applications." HardwareBee, 15 Feb. 2020, hardwarebee.com/fpga-common-applications/ (Accessed 3 July 2022).

[19] Technologies, GENERA. "FPGA Engineering: FPGA Applications - GENERA Technologies." Www.generatecnologias.es,www.generatecnologias.es/en/fpga_applications.html. Accessed 4 July 2022.

[20] DE2-115 User Manual. Terasic Technologies Inc, 2003-2013.

[21] 28BYJ-48 -5V Stepper Motor.

[22] Specifications Subject to Change without Further Notice. 4 Phase ULN2003 Stepper Motor Driver PCB.