STUDY AND DESIGN OF FPGA BASED FULLY CONNECTED NEURAL NETWORKS

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

VLSI DESIGN AND EMBEDDED SYSTEM

Submitted by:

KULDEEP SINGH BISHT

(2K20/VLS/07)

Under the supervision of

Asst. Prof. KRITI SUNEJA

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

MAY, 2022

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I KULDEEP SINGH BISHT 2K20/VLS/07 student of M. TECH (VLSI and EMBEDDED SYSTEM), hereby declare that the project dissertation report on **"STUDY AND DESIGN OF FPGA BASED FULLY CONNECTED NEURAL NETWORKS"** which is submitted by me to the Department of Electronics and Communication engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

31 MAY 2021 KULDEEP SINGH BISHT (2K20/VLS/07)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the project dissertation titled "**STUDY AND DESIGN OF FPGA BASED FULLY CONNECTED NEURAL NETWORKS**" which is submitted by KULDEEP SINGH BISHT, Roll No. 2K20/VLS/07 Department of Electronics And Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of technology, is record of the project work carried by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

31 May 2022 Asst. Prof. KRITI SUNEJA

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude and indebtedness to my highly respected and esteemed guide Ms. Kriti Suneja (Assistant Professor, ECE) for having suggested the topic of my project and for giving me complete freedom and flexibility to work on this topic. She has been very encouraging and motivating and the intensity of encouragement has always increased with time. Without her constant support and guidance, I would not have been able to attempt this project. I would like to thanks my parents, friends for their continuous support without them it would not be possible.

ABSTRACT

The project's goal is to design a field programmable gate array (FPGA) based fully connected neural network for handwritten digit recognition. The project involves training the network utilizing the modified national institute of standards and technology dataset (MNIST), as well as designing and simulating a fully linked deep neural network (DNN). Initially, small modules such as memory and activation functions were created independently to test the fundamental functionality of neurons. A multi-layer neural network was created to improve the neural network's overall accuracy by performing the identification of handwritten digits. The network inputs, weights, and outputs are all expressed in this paper using a fixed point representation format. On-chip memory is used to hard-code network weights. Moreover, pipeline registers are utilized to ensure that data flows without error between layers. Using a state machine to control pipeline register access and simultaneously process output from one layer to the next layer with valid output control. Finally, the neural network is assessed using the test bench and MNIST test dataset as inputs, with parameters such as accuracy and resource utilization calculated, which may vary depending on the configuration. The TensorFlow libraries are used for training the network. The whole hardware description of a neural network is written in Verilog. For simulation and implementation, Xilinx Vivado 2020.1 is used in this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Background

"Machine learning" [1] has been ubiquitous in several scientific domains and commercial applications in recent years and has produced excellent results. "Deep learning" [1] is a branch of machine learning that primarily studies deep neural networks, which are statistical frameworks. Deep learning has accelerated the advancement of machine learning and artificial intelligence. As a result of this, "deep learning has become a popular area in the study of research" [2]. "Deep learning includes a multi-layer neural network model to extract high-level features, which are a collection of low-level abstractions" [3], to resolve complex machine learning difficulties. The common networks that are widely used as deep learning neural frameworks are Deep Neural Networks (DNNs) and Convolution Neural Networks (CNNs). They have been shown to have outstanding capabilities when it comes to solving complicated machine learning problems like image and gesture recognition, speech analysis, etc.

An "artificial neural network (ANN)" [4] is a data processing system that attempts to effectively replicate both the structure and function of the human brain. All of the operations in ANN models, including data collection and processing, overall network architecture, number of hidden layers, correlation between different parameters, and weight-bias calculations, are decided using learning and training methods. Artificial neural networks, complex "patterns recognition" [5], understanding and "stock

performance forecast" [6], tracking and management, and voice and image recognition are only a few of the most recent advancements in AI research. [7].

The study of artificial neural networks and related machine learning techniques with more than one hidden layer is known as "deep learning" [8]. The term "deep" in deep learning referred to the number of hidden layers in a neural network architecture. A neural network that uses a deep learning algorithm is referred to as a deep neural network. It captures and converts features using a stack comprising several hidden layers of non-linear processing elements [9]. The information from the preceding layer is being used as an input for each subsequent layer. Deep neural networks have sparked a lot of interest and adoption over the last few years across a wide variety of industries as well as applications. Whenever "general-purpose processors are not feasible because of performance, semiconductor expense, or power usage restrictions" [10], the need of customized hardware architecture for deep learning models becomes required under certain practical recognition applications.

Although deep learning models have been successfully incorporated into software on general-purpose personal computers, hardware implementations of deep learning models on semiconductor materials face a number of challenges, together with incorporating the activation function, maintaining the network's weights, and choosing the appropriate data representation model for calculations.

The selection of appropriate hardware depending on that purpose is the first and most important stage in developing the ANN model. Artificial neural network frameworks can handle vast amounts of data while simultaneously solving complex problems. Both the scale and complexity of the model deployed for ANN applications, even with the resource needs, are enormous. The system also necessitates parallel or simultaneous data analysis. The most often used hardware for this purpose are described in, including CPUs, GPUs, Servers, Clusters, FPGAs, and ASICs, among others.

CPU: It has a low computing efficiency and is incapable of meeting real-time application needs. As a result; it was unsuitable for parallel processing.

GPU: It does parallel processing, but it consumes a lot of power. The GPU is a multi-core processor that is used for the ANN architecture's processing and training. It was designed for neural network training and validation since it can perform multi-million operations per second. GPUs require a lot of power, are large, and are not suitable for low-power embedded systems. Only single and double-precision floating-point data formats are supported by the GPU. Other data types, such as 8/16 bit fixed point or 1-2 bit binary data, are not supported. Hence, GPU was also not suitable for on-field implementation.

ASICs: ASICs are low-power devices that can be used to implement artificial neural networks. However, their main drawback is that they are customized for a single application and are not reusable across architectures. Because ASICs have a long development cycle, they aren't really a good choice for implementing ANN architecture.

FPGA (Field programmable gate array): FPGAs are notable for their reconfigurable design. They provide you with the option of reprogramming the network design several times. They are low power, enable parallel computation, have flexibility in design, and the development life cycle for FPGA is short, as illustrated in the study. After an investigation of various hardware for ANN model implementation, FPGAs were found to be useful for implementing ANN models in hardware. Following that, we'll go over the most typical ANN architectures for image applications.

1.2 Objectives of thesis:

The thesis major goal is broken into two parts:

1. Design part: The main focus of this section of the thesis is on developing neural network models and then implementing them on FPGA. The design part follows step1 to step 5 as shown in fig. The criterion for choosing proper neural network architecture is entirely dependent on its flexibility and the application that will be implemented using it. We divided the job into two domains after deciding on the architecture. The first domain will concentrate on model training and the generation of required data for evaluation. The

second domain will concentrate on developing layered neural network models from the bottom to top approach.

2. Evaluation part: Throughout this section, the model developed will be assessed with varying design elements. The two main areas of concern here are design accuracy in forecasting the correct image and resource utilization. We examine the effects of changing various design elements on neural network accuracy and FPGA resource consumption.

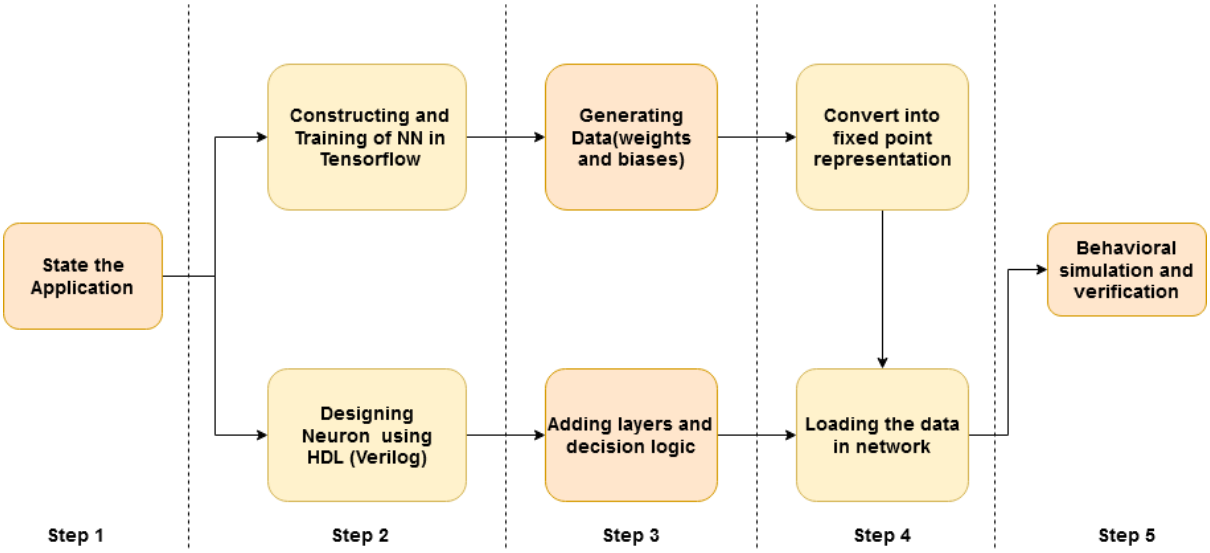

Fig 1.1: Outline of project

1.3 Outline:

The following are the chapters that make up this thesis:

Chapter 1, Introduction: In this objective of the thesis is introduced. Why FPGAs are the better choice for ANN implementation is explained. Also, the complete outline of the project in the flow diagram is provided.

Chapter 2, About Neural networks: To understand neural networks, basic information has been provided. This chapter covered the fundamental model of an artificial neuron, the basic unit for any neural network, and how a neural network works effectively for finding patterns in images. At last, some of their applications were discussed.

Chapter 3, Training and data generation: The essential part of the thesis, data generation, and processing, took place in this chapter. This will help in the simulation and validation of the design in FPGAs at a later stage of work.

Chapter 4, Hardware Implementation: In this chapter, first there is a brief introduction to FPGAs given then the design of neurons in FPGAs is discussed. Then layered architecture of the neural network is given. In the end verification of neural network has been discussed with the help of a training dataset.

Chapter 5, Results and Discussion: All the results are based on behavioral simulation and verification discussed here.

Chapter 6, Conclusions and Future Work: This chapter summarizes the conclusions of the thesis and addresses possible future research in the field.

Chapter 2

ABOUT NEURAL NETWORKS

2.1 Artificial Neuron

In 1943, neuroscientist Warren S. McCulloch and logician Walter Pitts developed the very first theoretical design of an artificial neuron. In their study report, they define a neuron as a single cell that receives, analyses, and produces information from a network of cells. In a neural network, the neuron, often called a node or a unit, is the basic computational unit.

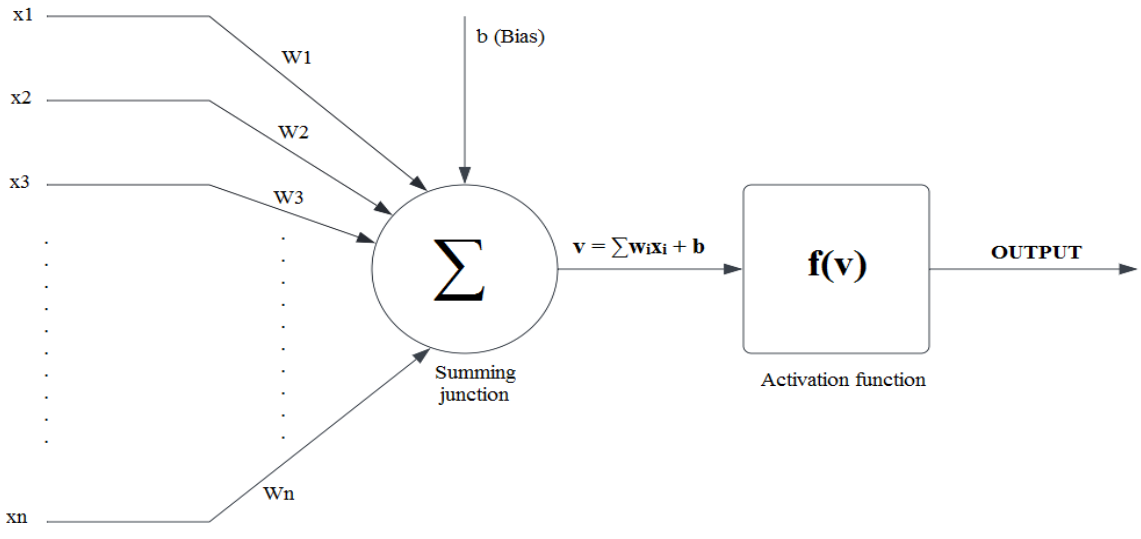

Fig 2.1: An Artificial neuron

It generates a signal by taking input from other neurons in the network or from an external source. Each input is assigned a weight (w), which is decided by its comparative relevance to other components. Synaptic strengths (w) are being learned to adjust both direction and the strength of an excitatory (positive weight) or inhibitory (negative weight) neuron's action on another. For the most basic model, every bit of information gets carried via neurons towards the summation junction, where it is summed. The neuron can fire and send a signal up to its axon if the total amount surpasses a particular threshold. In the mathematical computation, we assume that the exact durations within the peaks are irrelevant and that only the frequency of the firing communicates information. We use an activation function (e.g., a sigmoid function) to simulate the neuron's firing rate, which depicts the frequency of spikes along the axon.

2.2 Components of Artificial neuron

Inputs: Inputs are the data that we need to determine the output value. They could be considered features or attributes in a dataset.

Weights: Weights are real-world figures linked to every feature which demonstrates how significant a certain feature is in predicting the final value.

Bias: This is a neural net parameter that, in addition to the linear combination of both neuron's input data and weight, will be used to change its result. As a result, a bias value enables you to shift the activation function to left or right that might be useful for learning.

Summation Junction: The summation junction's role is to combine the products of inputs and their weights and finally evaluate their sum using the bias value.

Activation function: It's being used to make the system quite non-linear, allowing the neural network to perform complicated patterns on given information. A neural net is

little more than a linear model with no activation function. As a response, designers incorporate a non-linear transformation to the neuron's inputs, and we bring nonlinearity into the network via an activation function.

2.3 Neural network

A neural network, often known as an "artificial" neural network (ANN), is a computational model that is inspired by the way biological neural networks process information in the human brain. An artificial neural network (ANN) is a data processing system that seeks to mimic the design and function of the human brain. A neuron is the brain's basic computational unit. The human nervous system contains roughly 86 billion neurons that are coupled by approximately 10^{14}-10^{15} synapses.

Every neuron in a neural network is a processing element that generates an output by applying a function to its input (s). The architecture of a neural net is determined by the number of networks of interconnected groups of neurons that perform similar tasks, which is described by the layers of the network. In nature, a layer might be input, hidden, or output. A neural network may contain multiple hidden layers. The depth of the network is determined by the number of layers. All neural networks have a classifier layer that generates the network's final output (s).The number of outputs in such a layer is equal to the number of classes the network must anticipate.

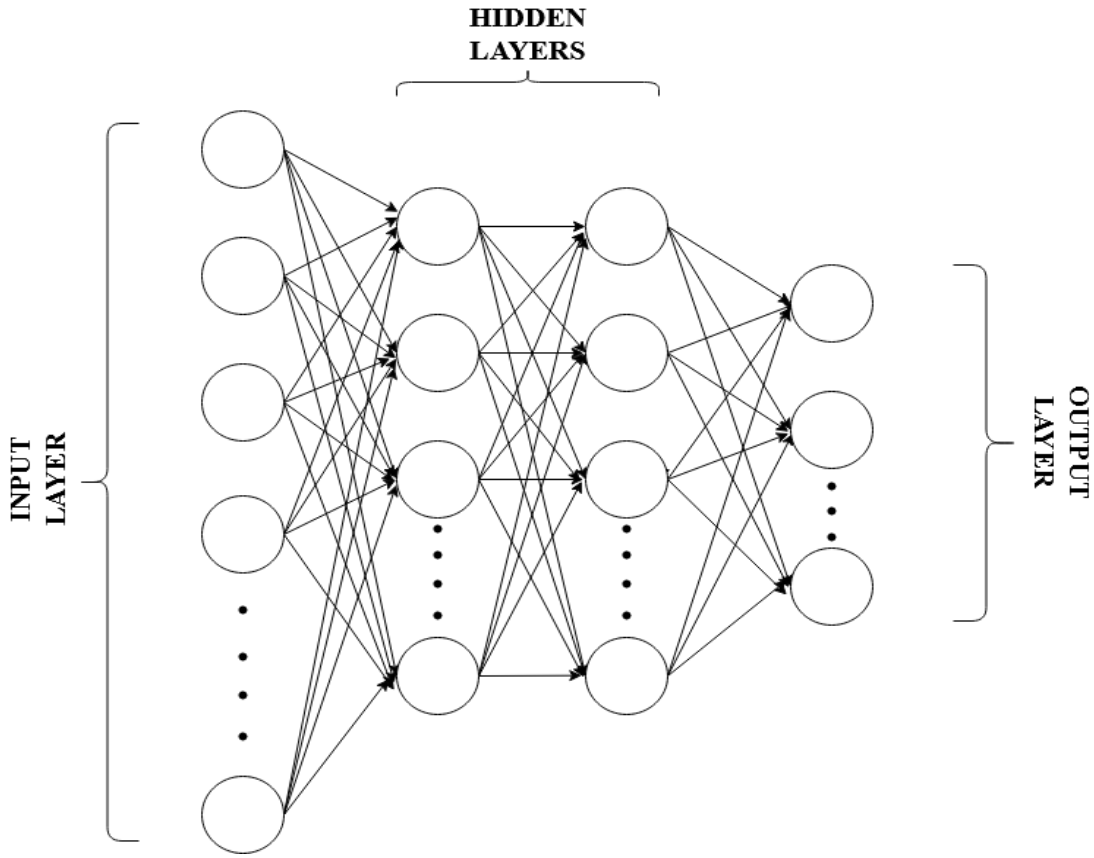

Fig 2.2: Fully connected neural network

2.4 Components of Artificial Neural Network

1. **The Perceptron:** A perceptron is a type of artificial neuron. A perceptron is the simplest neural network imaginable, a computational representation of a single neuron, and was invented in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory. A perceptron consists of one or more inputs, a processor, and a single output. The perceptron works on a "feed-forward" concept, which means that inputs are sent into the neuron, analyzed, and an output is produced.

2. **Input layer:** This layer does not do any computations instead, it just passes information to the next layer (hidden layer most of the time). A layer is a collection of neurons (perceptron).

3. **Hidden layer:** Hidden layers perform intermediary processing or computations before transferring the weights (signals or information) from the input layer to the next layer (another hidden layer or to the output layer). The number of hidden layers in any neural network is completely dependent upon its application as well as its design. It is feasible to build a neural network without a hidden layer.

4. **Weighted connections:** The network is made up of connections, each of which transfers the output of one neuron to the input of the next. Weights are the true values associated with each feature, and they indicate how important that feature is in predicting the final value.

5. **Learning Rule:** A learning method or process is a procedure that modifies all variables around a neural net over a certain input in order to produce a desired output. A learning process is like an arithmetic computation, which can be implemented using mathematical logic. By implementing such a procedure throughout the neural net, it will improve performance in terms of accuracy. It is a common characteristic of most learning methods to adjust the weights and bias values.

2.5 Types of neural network

The ANN network structure should be simple and straightforward. Recurrent and non-recurrent structures are the two most common forms of arrangement. The auto-associative or feedback network is also known as the recurrent structure, while the associative or feed-forward network is known as the non-recurrent structure. In a feed-forward network, the signal travels in only one direction, but in a feedback network, the signal travels in both directions by inserting loops into the net.

1. Feed Forward Neural Network

A feed forward neural network [11] is an artificial neural network with connections that do not form a cycle between the units. The information in this network only flows in one direction: forward, from the input nodes to the output nodes, going through any hidden nodes (if any). In the network, there are no cycles or loops.

1.1 Single-layer perceptron

This is the simplest feed-forward neural network, as it has no hidden layers and only one layer of output nodes. The input layer is excluded from the count of layers since no computations are performed there; instead, the inputs are sent straight to the outputs via a series of weights.

1.2 Multi-layer perceptron

Multiple layers of processing units are interconnected in a feed-forward manner in this type of network. Each neuron in one layer is connected to the neurons in the next layer via directed connections. The units of these networks use a sigmoid function as an activation function in many applications. In contrast, MLPs are more beneficial for a variety of reasons, one of which is their ability to learn non-linear representations (most of the time, the data presented to us is not linearly separable). They are sometimes called fully connected neural networks.

1.3 Convolution neural network

Convolutional neural networks, like conventional neural net models, contain neurons with learnable weights and biases. They're multilayer perceptrons with minimal preprocessing requirements. The convolution layer, which takes up the majority of the network's time and is only effective when nonlinearity and pooling layers are included [12], is the most important in a CNN. CNNs use two types of layers to extract information from input features: convolutional and pooling layers. Among the various applications are image and video recognition, reinforcement learning, and natural language processing. Each neuron within those layers gets associated with a unique area of the image to which it applies convolution or pooling. The network is not fully connected due to this division into regions. Sparsely connected networks are another name for them. The training of CNNs necessitates a vast amount of data.

2. Recurrent neural networks

A directed cycle is formed by the connections between units in a recurrent neural network (RNN). As a result, the system's dynamic aspects could be displayed. They are utilized to comprehend material that is temporal or sequential. Recurrent neural networks typically involve memory as a component. Unlike feed-forward neural nets, RNNs utilize their internal memory to analyze arbitrary sets of data. RNNs utilize different sets of data in a sequence to develop improved predictions. These are normally recommended when the output is dependent on a number of inputs. They accomplish this through receiving input and affecting output by looping the activations of previous or subsequent nodes in the sequence. As a result, jobs like unsegmented, connected handwriting identification, speech recognition, and other general sequence processors can benefit from them.

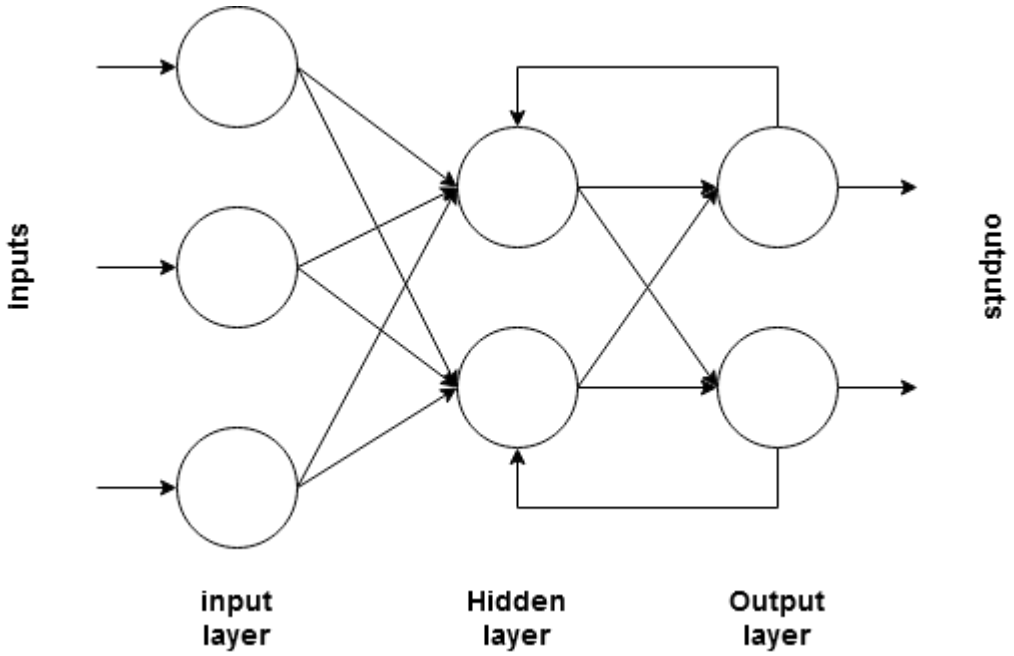

Fig 2.3: Recurrent neural network

2.6 APPLICATIONS

Pattern Recognition: In terms of functionality, ANN provides a framework for pattern recognition that demands huge networks of nonlinear and simple components known as neural nets. In many problems, ANNs are becoming more interesting, efficient, effective, and successful in pattern recognition [5]. Facial recognition, optical character recognition, and other similar technologies are examples.

Time Series Prediction: Predictions can be made using neural networks. For example, in stock market forecasting, every day the activities of the stock market are often exceedingly complicated.

The number of factors determines if a share will either climb or decline. With so many uncertainties in the market, neural nets would gather as well as analyze enormous quantities of data efficiently. We may, therefore, use them to forecast stock prices. Is the stock price going to rise or fall tomorrow? Will the stock price increase or decrease tomorrow? Or weather: is it going to rain or will it be sunny? or predictions about weather.

Signal Processing: Signal processing is required by cochlear implantation and hearing aids help in filtering away unwanted frequencies while magnifying the actual relevant frequencies. Neural networks are sometimes developed to interpret and filter audio signals.

Control: The latest developments in self-driving car research have just caught our attention. When it comes to physical car driving judgments, AI (deep learning models) has been successfully used.

Soft Sensors: A sensor is a device that analyses large amounts of data. A thermometer is a sort of sensor that measures air temperature; similarly, there are many sensors available that detect changes in the environment. We may also monitor pressure, humidity, air quality, density, and other parameters in addition to temperature. Information from a large number of discrete sensors can be analyzed and evaluated using neural net models.

Error Detection: Neural networks may be trained to generate an output whenever anything goes terribly wrong since they are designed to be efficient in the field of detecting and making predictions. Consider a long-running neural network that has been monitoring one's everyday actions. After learning the patterns of behavior, it may be able to alert you when something is wrong.

Medical Diagnosis: Biomedical analysis, drug development, diagnostic systems, image analysis, and other areas of medicine have all seen success with artificial neural networks. Artificial neural networks play an essential role in image analysis as well, where they are used in conjunction with digital image processing for recognition and classification. CNNs are being utilized throughout the health sector with ultrasound, CT scans, MRI, and X-ray detection. Using artificial neural networks, it is feasible to monitor a range of benchmarks related to health, including oxygen consumption, pulse rate, and sugar levels, as well as predict the patient's response to treatment..

Chapter 3

TRAINING AND DATA GENERATION

The main agenda for this section is to find out about training of neural networks and gathering data for later stages. According to how they learn, neural networks can be divided into three categories:

Supervised learning: The purpose of supervised learning is to find functionality for a training set comprising a given set of input and output pairings that accurately reflects its actual function. In this situation, the output response from every input is already known with high accuracy, and the labeled instances in the training data set are used to monitor the training process.

Semi-supervised learning: Semi-supervised neural networks have a high number of unlabeled samples and few labeled examples. Using both types of examples, semi-supervised NNs find their optimum classification strategy for given data.

Unsupervised learning: Unsupervised neural networks, apart from the above two, learn correlations from data received without acquiring regular feedback; they're commonly employed in classification applications.

3.1 MNIST Dataset

In 1998, the United States Census Bureau created the MNIST (mixed national institute of standards and technology database) [13]. The MNIST dataset contains a number of

70000 samples, 60000 of which are for training and the rest are for testing. The database is made up of two origins. NIST's special dataset 1 gathered from high school pupils and stored in NIST's special dataset 3 information obtained from census bureau personnel. The training and test sets were chosen to avoid having the same person work on both. And over 250 writers' samples are included in the training set.

Fig 3.1: MNIST data samples

3.2 Loss function

In supervised learning, training neural networks comprises modifying the network weight and bias parameters and minimizing the overall prediction error. Whenever the design with acquired data gets close to an accurate response, the loss function is the amount of utility lost. In other words, the loss function evaluates whether the whole anticipated model is correctly specified in terms of fitting the data.

3.3 Back propagation algorithm

The error "back-propagation algorithm" [14] is the most frequently used approach for training ANNs. After arbitrarily allocating the network weights, the back propagation algorithm determines the necessary corrections [14]. The artificial neural net layers are connected by two feed-forward and two back-propagation routes. In the feed-forward path, this same neural net first receives input values from either a particular training data set, a group of data sets, or all training data sets. Those same input data are combined first by initialization weighting factor, after that addition of all weighted input are computed as well as sent across each predefined activation function until they reach the output layer. The weights and bias values remain constant throughout the feed-forward process.

Once the outputs have arrived just at the final (output) layer, the "loss function" [14] gets determined by subtracting the output layer response from the predicted output value. In the back-propagation path, the loss function value gets propagated through the neural network multiple levels and adjusts the weights as well as bias parameters.

3.4 Gradient descent

The “gradient descent algorithm” [15], a first-order approximation algorithm that updates the weights of a given model, is among the most commonly used neural net optimization techniques. This technique will generate the loss function derivative for each neuron in each layer about each weight and bias value. The learning rate and gradient values are used to adjust the model's weighted vector and biases. The feed-forward and back-propagation paths are then done several times (epochs) until the model has been trained as per the training algorithm's constraints. "Stochastic gradient descent, or SGD, is a gradient descent-based optimization technique that is more time-efficient." [15]

3.5 Training

Before training the neural network, first need to design a framework neural network using TensorFlow. TensorFlow is an open-source platform help in implementing deep learning and machine learning applications. As it is written in python programming language so it is considered simple to understand. To build this neural network framework, we use Google Colab, a cloud-based Jupyter notebook that is free to use. To use this any kind of setup is not required which is the great thing about it.

Steps to create fully connected neural network:

1. Loading the data and preprocessing:

In this step the first thing is to import libraries and models necessary to build models. Then load the MNIST dataset using TensorFlow keras library. At the end it is important to divide the imported data set into training and validation sets. And then we do normalization on both train as well as the test set.

```
#import models and libraries
import tensorflow as t
import json as js


#loading tha MNIST data set
mnist = t.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

#Normalization of dataset (between 0 to 1)
x_train = t.keras.utils.normalize(x_train, axis = 1)
x_test = t.keras.utils.normalize(x_test, axis = 1)
```

2. Definition of neural network:

In this step construction of the neural network model takes place. Here the first nature of neural network takes place i.e., sequential or functional. Then layer formation takes place in which layer properties like numbers of neurons, sparsely or dense, activation function.

```
# definition of Neural network model

model = t.keras.models.Sequential()
model.add(t.keras.layers.Flatten())
model.add(t.keras.layers.Dense(30, activation=t.nn.sigmoid))
model.add(t.keras.layers.Dense(30, activation=t.nn.sigmoid))
model.add(t.keras.layers.Dense(10, activation=t.nn.sigmoid))
model.add(t.keras.layers.Dense(10, activation=t.nn.softmax))

```

3. Compile the model:

This step of compiling the model is final point in creation of neural network model. After this, the model is ready for training. In this we specify matrices, optimizer and loss function which are important features for model to produce its final output.

```
#compilation
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
```

4. Training the model:

The above model was trained with the training data set. First, a training set is added to the design, and then another parameter called epoch is introduced to improve the accuracy of the proposed design. An epoch is a basic criterion that specifies how many times the learning technique is run on the entire training data set.

```
#Training the model
model.fit(x_train, y_train, epochs = 10)
```

5. Evaluating:

And finally the evaluation phase where to check the model is the best match for the given problem or data set.

```
#Evaluating the model
val_loss, val_acc = model.evaluate(x_test, y_test)
print(f"Loss = {val_loss}")
print(f"Accuracy = {val_acc}")

```

```
Epoch 1/10
60000/60000 [==============================] - 96s 2ms/step - loss: 0.5811 - accuracy: 0.8352
Epoch 2/10
60000/60000 [==============================] - 97s 2ms/step - loss: 0.2284 - accuracy: 0.9378
Epoch 3/10
60000/60000 [==============================] - 96s 2ms/step - loss: 0.1748 - accuracy: 0.9516
Epoch 4/10
60000/60000 [==============================] - 94s 2ms/step - loss: 0.1471 - accuracy: 0.9586
Epoch 5/10
60000/60000 [==============================] - 94s 2ms/step - loss: 0.1261 - accuracy: 0.9646
Epoch 6/10
60000/60000 [==============================] - 93s 2ms/step - loss: 0.1137 - accuracy: 0.9673
Epoch 7/10
60000/60000 [==============================] - 94s 2ms/step - loss: 0.1021 - accuracy: 0.9707
Epoch 8/10
60000/60000 [==============================] - 102s 2ms/step - loss: 0.0948 - accuracy: 0.9723
Epoch 9/10
60000/60000 [==============================] - 100s 2ms/step - loss: 0.0878 - accuracy: 0.9743
Epoch 10/10
60000/60000 [==============================] - 96s 2ms/step - loss: 0.0832 - accuracy: 0.9757
313/313 [==============================] - 1s 1ms/step - loss: 0.1562 - accuracy: 0.9589
Loss = 0.15619273483753204
Accuracy = 0.958899974822998
```

3.6 Generating data

This is the most crucial step after neural network construction and training in TensorFlow. After training the weights and biases of every neuron in each layer has been updated so to extract those weights and biases, we write an additional python script. Using these weights and biases values in validation of neural network design in hardware.

```
# Generation of weights and biases
list_weight = []
list_biase = []
for i in range(1,len(model.layers)):
    weights = model.layers[i].get_weights()[0]
    list_weight.append((weights.T).tolist())
    bias = [[float(b)] for b in model.layers[i].get_weights()[1]]
    list_bias.append(bias)

data_weight = {"weights": list_weight}
f_w = open('weights.txt', "w")
js.dump(data_weight, f_w)
f_w.close()

data_bias = {"biases": list_bias}
f_b = open('biases.txt', "w")
js.dump(data_weight, f_b)
f_b.close()
```

3.7 Data Representation

Before heading to implementation in FPGA, first there is a need to represent the data generated during the training phase so that it can be utilized in later stages of work. The weights and bias can be positive or negative real numbers i.e., have both integer part and fraction part. “Fixed-point and floating-point arithmetic are two main forms of arithmetic representation”. [16] In FPGA, a fixed-point number has a predefined set of numbers before and after the decimal point as in fig. The integer and fractional parts of the floating point do not have a set number of digits allotted for them. A fixed-point number is represented as an integer that has been multiplied by an implicit factor. When compared to floating-point arithmetic, fixed-point arithmetic is extensively employed in FPGA based algorithms since it runs faster and consumes fewer hardware components. However, one disadvantage of fixed-point arithmetic is that the user must estimate the data range and select the scaling factor based on the fractional part size, making the system more prone to errors. In FPGAs, fixed-point seems to be the “most useful approach for increasing speed and reducing area” [16].

Fixed point format is easier to express and compute, but it may result in a little loss of overall accuracy. That's the reason why fixed-point representation is employed in this work to represent the data obtained from training part.

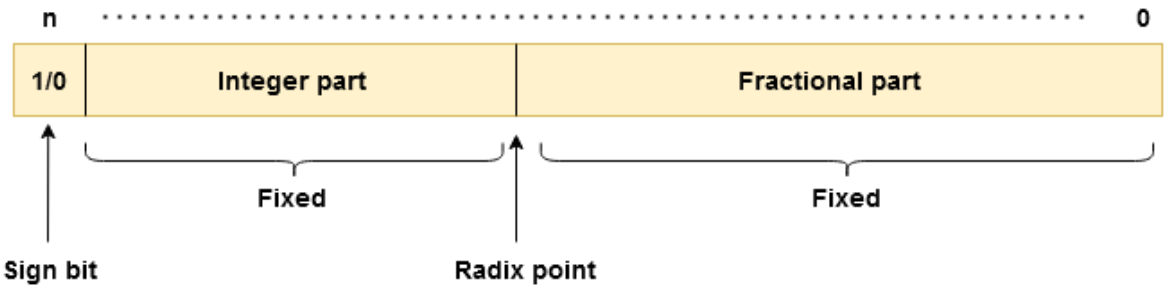

Fig 3.2: Fixed-point representation

Chapter 4

NEURAL NETWORK IMPLEMENTATION

4.1 FPGA Overview

"FPGAs (Field Programmable Gate Arrays)" [17] are solid state devices that may be programmed electrically to build any digital circuit. An FPGA is made up of a series of programmable logic blocks connected by programmable interconnects. They differ from Application Specific Integrated Circuits (ASICs), which are custom built for a specific design because of their reconfigurability. FPGA designs are modeled using Hardware Description Languages (HDLs) such as Verilog and VHDL, just like ASICs. The HDL design is then built and implemented to create a configuration file, often known as a bit stream file. This bit stream file describes how different FPGA components should be linked together. The design is programmed to continue until the FPGA is powered down after this bit stream file is downloaded to the FPGA.

FPGAs have been a common choice for engineers to prototype ASICs and SoCs designs to test various parts of design since they were first introduced. Their reconfigurability is the fundamental reason for this. If a flaw is discovered in the design, it may be fixed by simply altering the HDL code and downloading fresh bit streams into FPGAs. FPGAs can always keep up with future changes because they are re-configurable. Another significant benefit of FPGA is the reduced time required to fully construct a workable design. On FPGAs, a design can be made functional and evaluated for many scenarios without the lengthy fabrication process. Because ASIC designs are designed for a specific design, FPGAs have some disadvantages over ASICs, such as their reconfigurability,

which makes them slower in clock speed than ASICs to find the most efficient routing and connectivity for that design. In comparison to custom ASICs, FPGA designs consume more power. FPGAs are limited in their capabilities. FPGAs are preferred for low-speed designs and low-volume production in general. FPGAs also contain unique hardware built in, such as block RAMs, distributive memory digital clock management, high-speed I/O, soft-core embedded CPU, DSP blocks, and so on, that may be utilized to create nearly any form of design.

4.2 Memory

There are two types of memories in FPGAs: Block RAM and Distributive RAM/ROM [18]. Distributive RAM is synthesized by utilizing LUTs inside FPGAs, while block memory is generated by cascading or stacking a large number of registers. Both memories have distinct applications and are suitable for specific areas of interest. As a result, depending on the area of interest, we can implement memory using either block RAM or distributive RAM. Block RAM has some advantages to distributive RAM, such as the ability to hold a huge quantity of data and the ability to be faster than the latter. Furthermore, distributive RAM uses more LUTs than block RAM. Because this project involves a huge amount of data (weights and biases), it is preferable to use block RAM. By employing control signals to enable the memory, data can be read or written. Both memories write synchronously, but read asynchronously in Block memory and synchronously in Distributed RAM. As design, the synthesizing program chooses appropriate RAM depending on heuristics which thus result in the best for most designs, and found in the resource use report generated after implementation.

The two conventional methods are used to initialise memory blocks:

(1) Using Verilog/VHDL configuration files to provide directly the memory beginning values.

(2) Using an additional file to provide the RAM beginning values.

So, in general, we employ an approach to provide memory with initial values that may be read from an additional file holding data values generated during training in the form of MIF. This file is used as an input file for memory initialization in the simulator. The MIF file has a text format and includes data for every memory element. Since in this work a pre-trained neural network designed on an FPGA, the memory initialization concept comes very handy. In this work since fully connected neural network design is taking place, we use Block RAM memory to represent weighted interconnection between neurons. Also, in this work we implement activation function (sigmoid) using Memory blocks.

4.2.1 Weight and Bias memory:

Each neuron will consist of a memory which holds the weights for every other neuron from the previous hidden layer and bias memory for biases. These weights and biases are being calculated during designing and training of neural network using TensorFlow in chapter 3. We use memory initialization concept for specifying Memory initial content. After training of particular neural network in using TensorFlow libraries we get weights and biases corresponding to each neuron. We convert those weights and biases to binary values using fixed point data representation of certain data width.

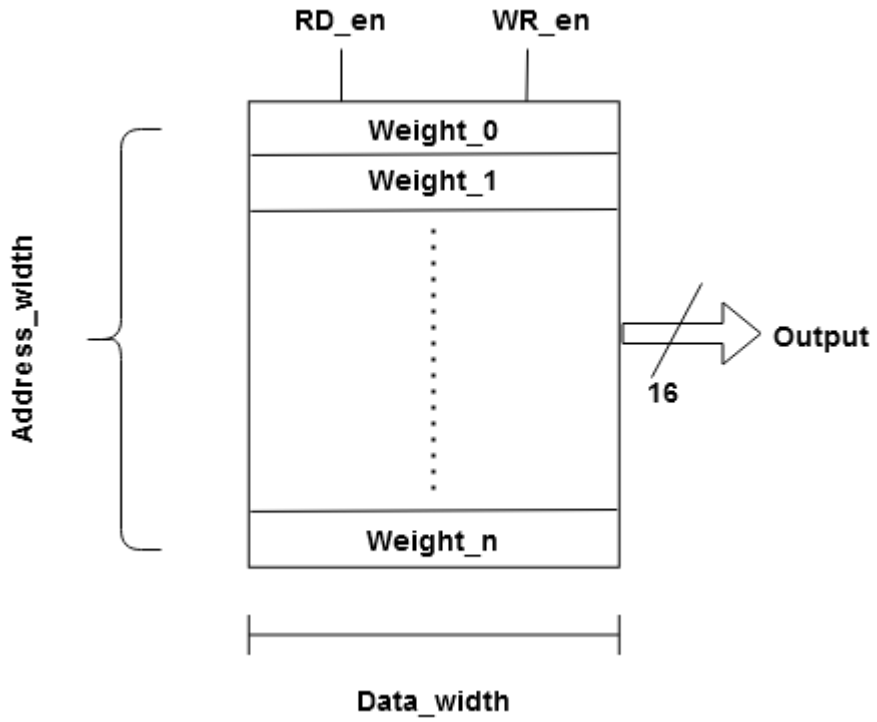

Fig 4.1: Weight memory

The size of weight memory in neurons at a particular layer is depending upon the number of neurons in the previous layer. Since it will be a fully connected layer every node in the same layer has the same number of inputs so they all have the same size of weight memory.

Size of weight memory = Address width x Data width

Address width = (n) , here n denotes the number of neurons in the preceding layers.

Whereas, Address width is calculated for each layer using the number of neurons in the preceding layer and width of data is fixed. Throughout this project, 16-bit data width was used to represent weights and input data. Initially, the memory contents were added using MIF files. When neural network starts the computations and read signal (RD_en) signal activates the neuron starts fetching weights to corresponding inputs from previous layer.

Bias memory is nothing but a simple register which initializes with bias for a particular neuron at a particular layer during training and learning in python.

4.2.2 Designing of Activation function:

Selecting the suitable activation function as well as building logic for its approximation to enhance the overall processing accuracy of a single neuron is one of the essential issues to consider before working on developing a neural network. The two activation functions that are widely used are:

1. Sigmoid

Since non-linearity is required in the model to achieve the accuracy in making the predictions therefore sigmoid is one of the commonly employed activation functions as it has non-linear nature. This function ranges the values in between 0 and 1. Mathematically defined as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

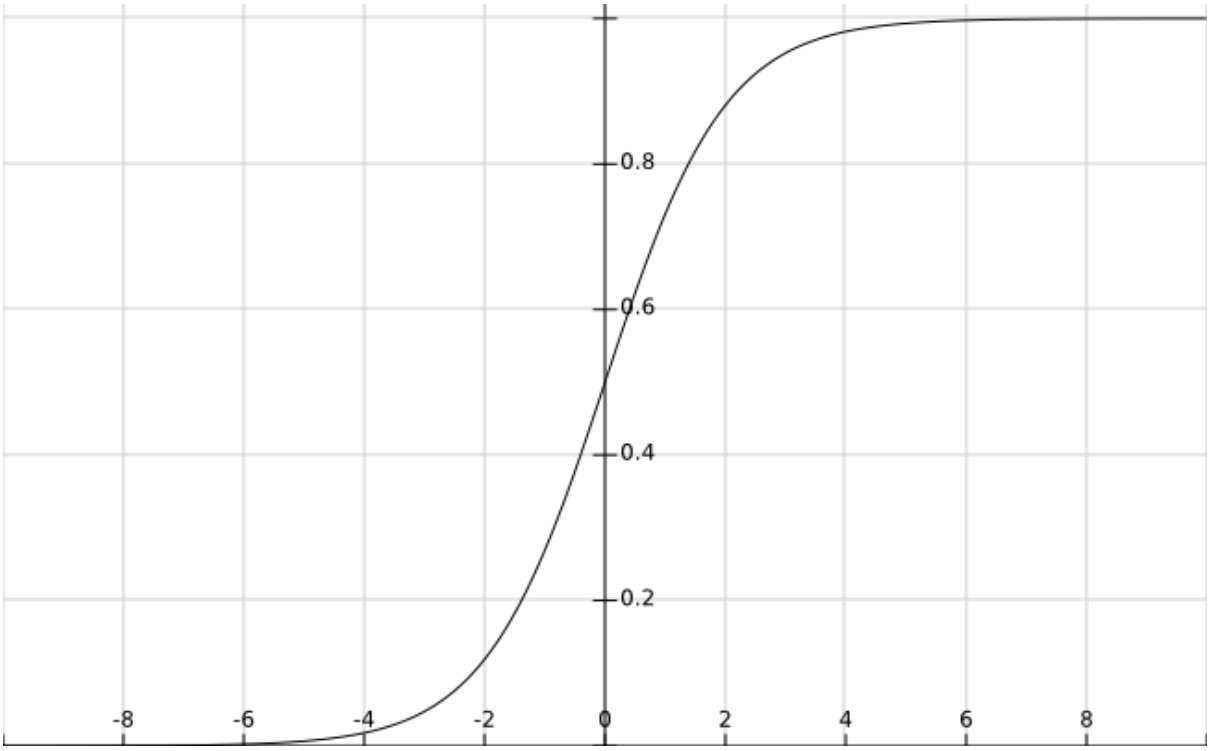

Fig 4.2: Sigmoid activation function

The graph of the sigmoid function shown in the fig 4.2 above has a persistent S-shaped arc including a region that is always differentiated. The function's derivative is:

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

Or,

$$f'(x) = f(x)(1 - f(x))$$

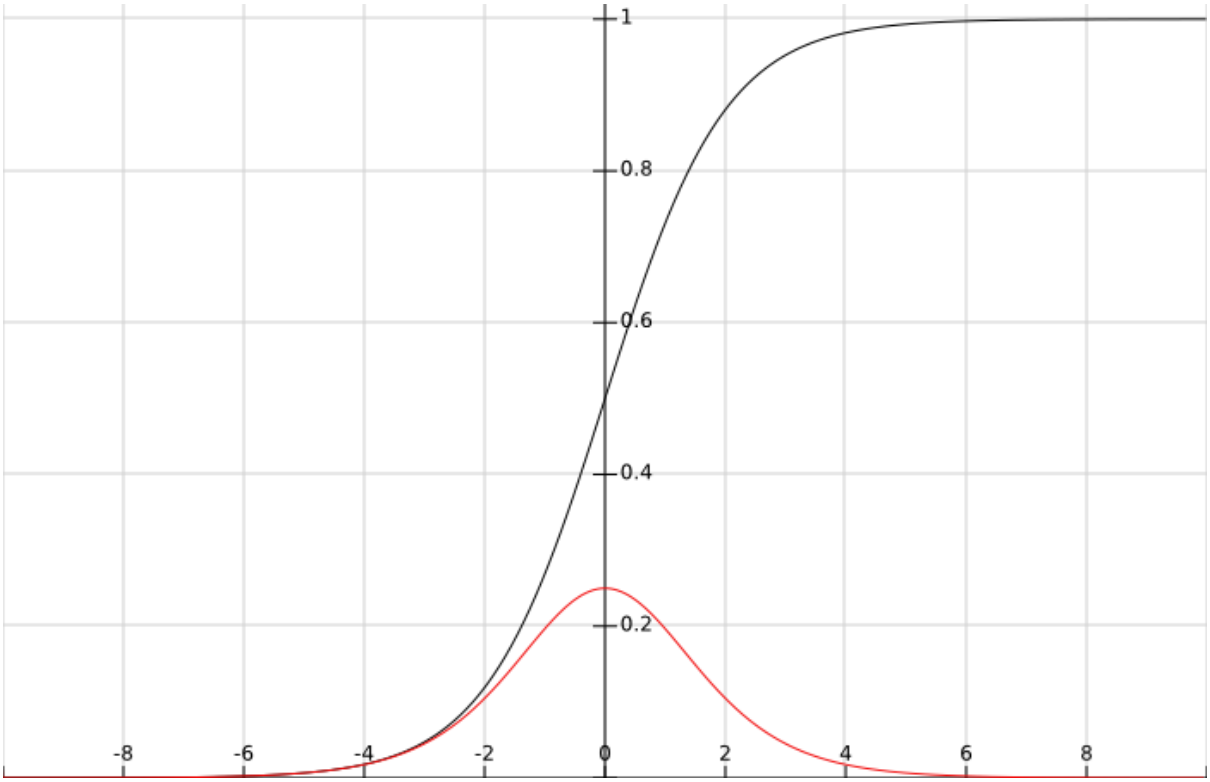

Fig 4.3: Sigmoid with its derivative

2. ReLU

The acronym for ReLU is rectified linear unit, non-linear function in nature employed in neural net models. Not only its implementation is simple but use of ReLU has upper edge of not activating all nodes simultaneously. When the result of the linear combination equals zero, this signifies how a neuron would be discontinued to process. It is mathematically defined as:

$$f(x) = max\ (0, x)$$

Even though a small number of neurons get triggered at a time, ReLU is quite effective than other functions since not all neurons are triggered simultaneously.

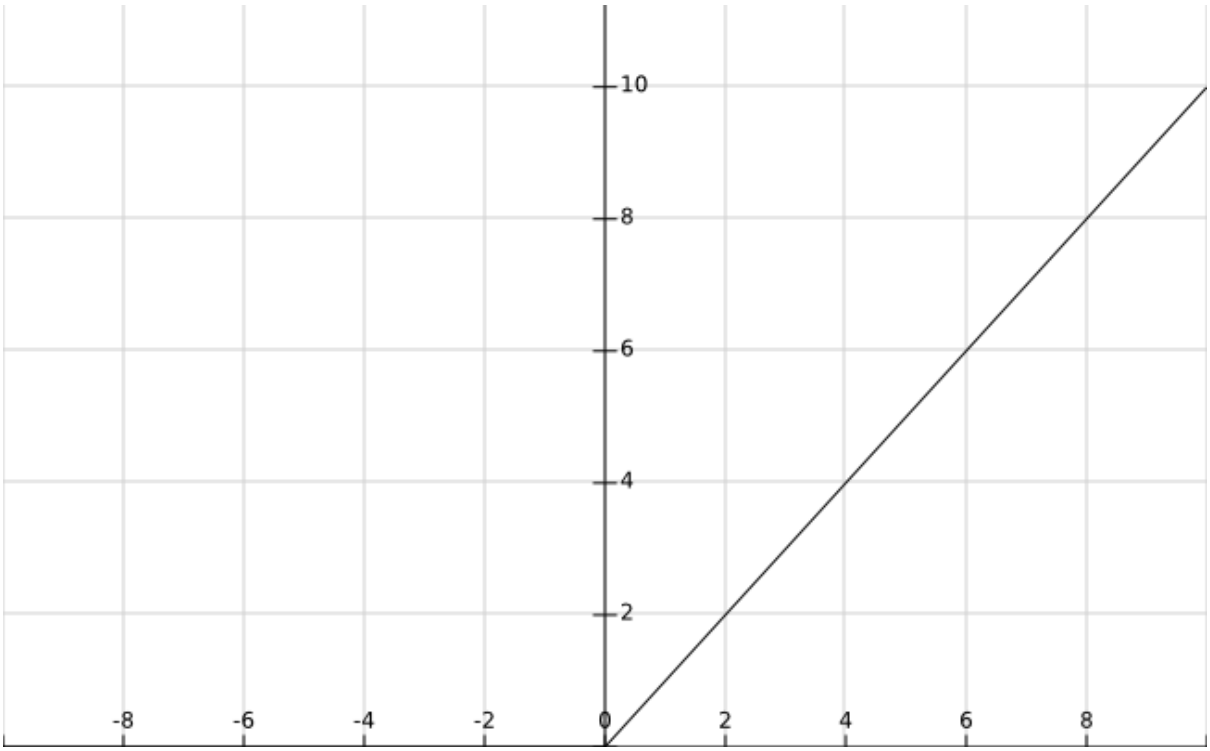

Fig 4.4 ReLU activation function

To approximate an activation function of any shape, activation function values are uniformly sampled and those values get stored in block memory (ROM) in FPGAs. The size of memory depends upon the total numbers of input samples and size of data width. The address field consists of input samples represented by fixed numbers of bits. The activation values of those input samples are pre-calculated and converted into binary of certain width. Suppose if input samples are represented by 8 bits and data width is 16 bits, then size of ROM is 2^8x16. There are a total 256 locations mean for this many input samples activation function get approximated. The approximation of any kind of activation function depends majorly on the numbers of bits used to represent input samples. More number of bits means more number of samples that would accurately represent the activation function. In fig, sigmoid function in range of [-6, 6] implemented using 2^8x16 ROM shown.

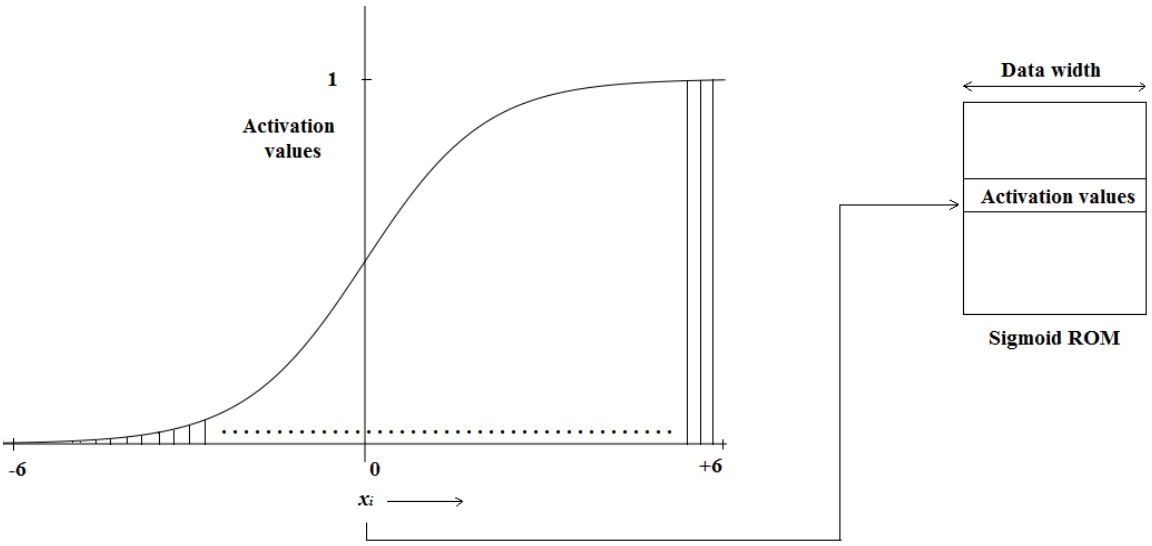

Fig 4.5: Implementation of sigmoid using ROM

4.3 Neuron Architecture

After designing the activation function, now the main job is to use weight and bias memory blocks to implement the mathematical expression for neuron. The neuron is smallest computational unit in neural network. Every neuron gets data from either previous layer neurons or from the input data set, it performs a dot product between the input and its corresponding weighting factor, inserts required bias values, and incorporates a non-linear activation before sending the output to other neurons. Mathematically a neuron action can describe by,

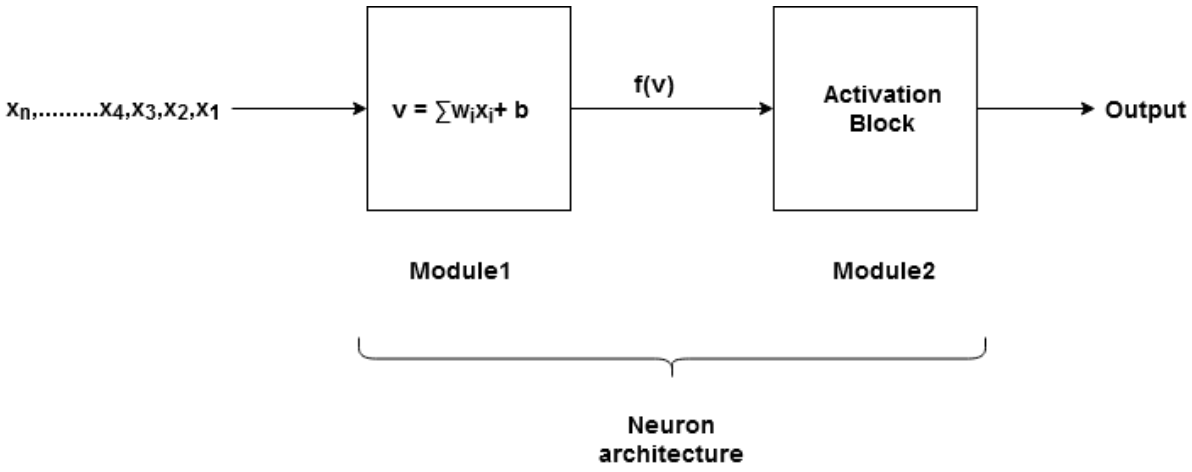

Fig 4.6: Mathematical model of neuron

The input applies to a neuron in a serial manner. In Module1 successive dot products take place between inputs and their corresponding weights. Then the whole dot products of input and weight along with the bias value are summed together using a summing junction. So to design Module1 we require weighted memory, bias register, MAC (Multiply and accumulate) unit and Summing point.

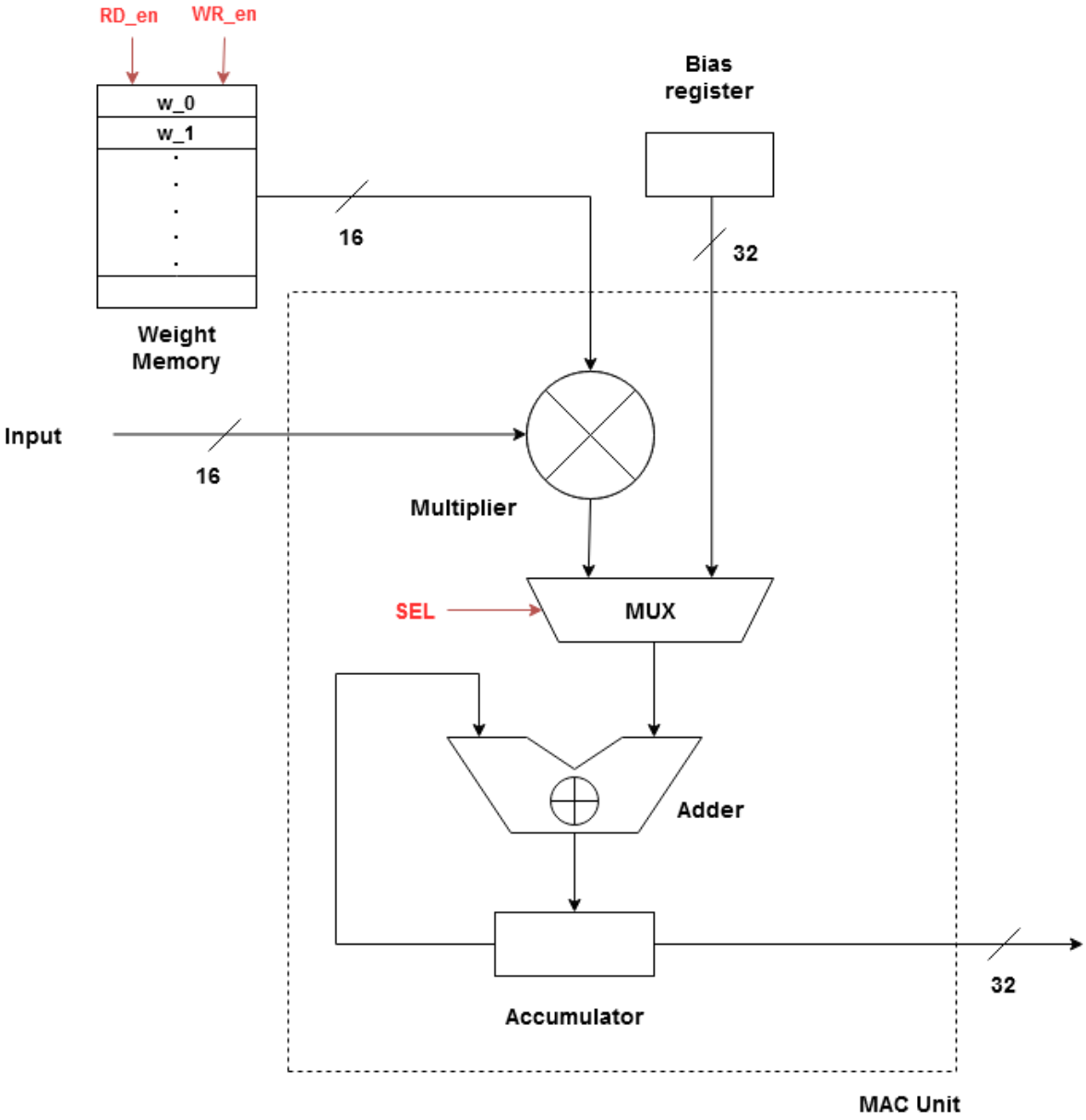

Fig 4.7: Module 1 architecture

As inputs are coming in serial fashion, the MAC unit takes input and weight value one by one, with reference to the clock, performs multiplication, and adds to the accumulator content. Initially, the accumulator is assigned zero bits; this will help us do successive addition on inputs and weights until the last input. Once the last input and its corresponding weight are added, a control signal gets triggered, which controls over operands for addition. For control over the operands of an adder, we use a simple MUX. The adder has two operands; one is an accumulator which is fed back and the other is

from MUX. This MUX helps us to select between product terms and bias content. Until the last input, this control signal is active low.

As soon as the last input arrives and is added with the accumulator, the signal becomes active high. So in the next cycle, bias content will be selected and added to the accumulator to achieve the operation of the neuron i.e.

$$v = \sum w_i x_i + b$$

We move on to Module2, which is another memory block used to approximate the activation function, after designing Module1. For a range of its input, the memory block was initialized with discrete values of the activation function output. The accumulator's output serves as an address field for the activation function ROM and generates the appropriate output. The Activation ROM will read only when the RD signal gets active high and this will happen when whole mathematical expression calculated by Module1.

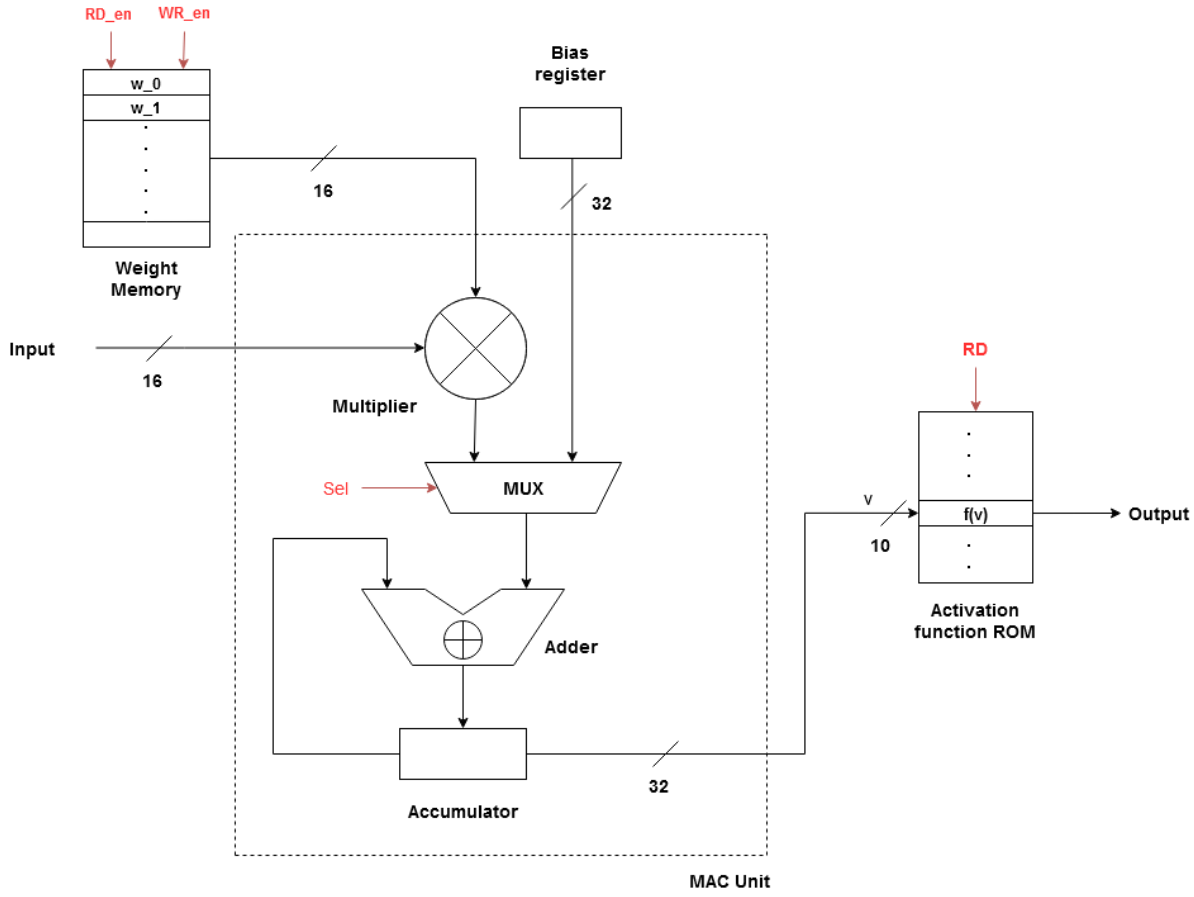

Fig 4.8: Complete model of neuron

4.4 Constructing neuron layers and pipelining register

The above neuron architecture serves as a basic unit for building neural networks. Calculate the number of neurons required in each layer before beginning layer construction. To build a neural network of size 120-30-20-10, for example, the first layer (input) has 120 neurons, followed by 30 neurons in the second layer (hidden), 20 neurons in the third layer (hidden), and 10 neurons in the fourth or last layer (output). Because the neural network is developed for an application that requires MNIST datasets, the default number of neurons in the input layer is always 784 and the output layer is always 10. The neural network configurations 784-200-10, 784-150-50-10, and so on are suitable for usage with the MNIST dataset.

The key concerns now revolve around how to ensure that communication across layers is reliable and error-free. Every neuron in this design receives inputs from prior layers at the same time, implying that each neuron is fully connected to every other neuron in previous layers. As a result, adding a shift register between layers to hold data from all the neurons in the preceding layer is the solution to this problem. The number of neurons in the previous layer and the data width determine the size of the shift register. The shift register in fig 4.9 will shift the data width amount of data to all neurons in the next layer with each clock pulse. So that each clock pulse gives the same input to all neurons in the next layer.

Here, pipeline action is employed by shift registers to prevent bogus information from propagating forward and all the neurons at every layer process simultaneously.

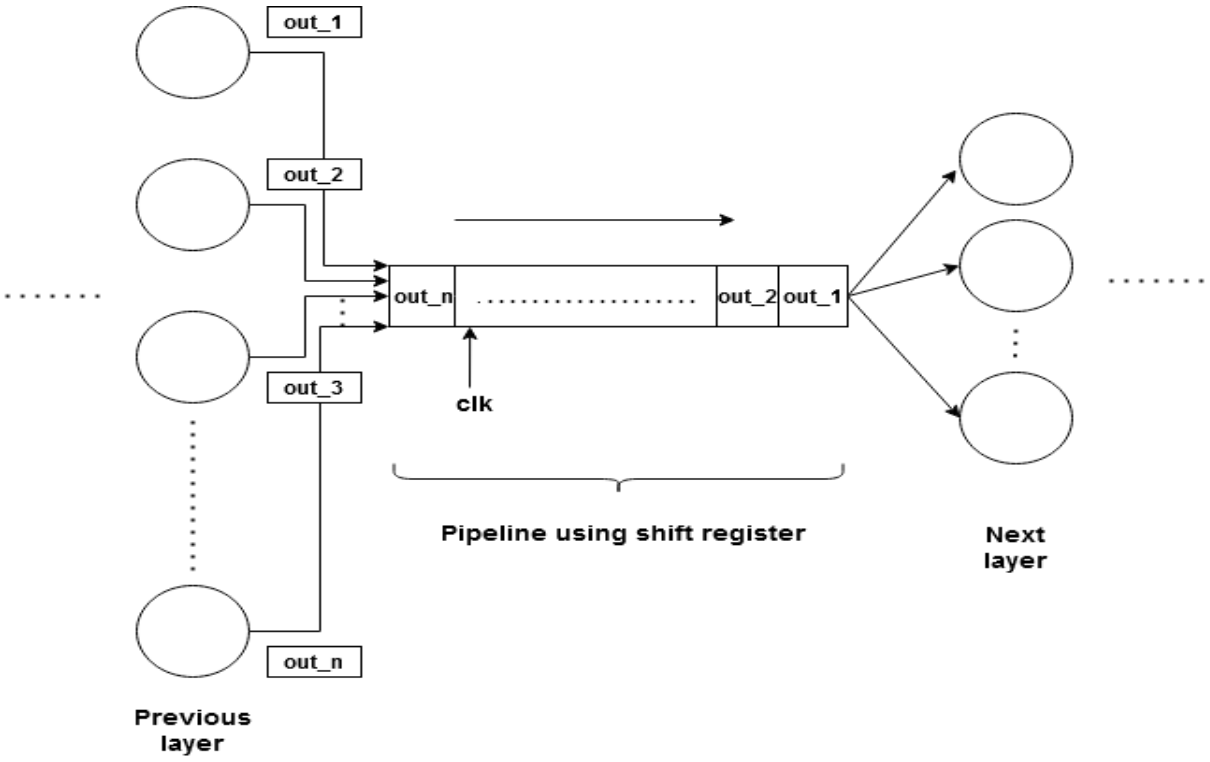

Fig 4.9: Introduction of Pipeline using Shift register

4.5 Verification of neural network with MNIST dataset

The MNIST dataset has around 60,000 datasets for training purposes and 10,000 for testing or validation purposes. The data consists of images of handwritten digits (0–9) in size 28 x 28 pixels. This total of 784 pixels data acts as input to the input layer, i.e., 784 neurons. These test data sets are already converted using fixed point in binary form with 16 bits as the data width. Along with the data set, the expected value of the input is also inserted into the design for verification at the end of every computation. At the end, the output layer of neural network decision logic has been used. The decision logic decides which neuron output to send for comparison with the expected value. The output will be compared with the expected one, and based on this; we later compute the accuracy of the model.

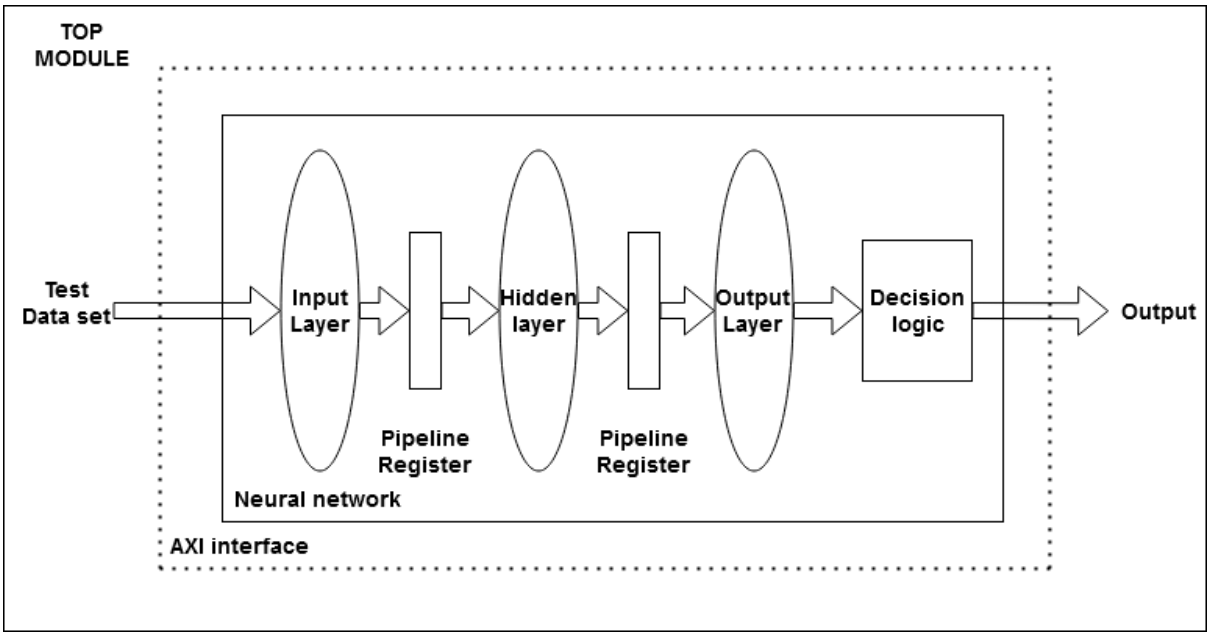

Fig 4.10: Complete design with Top module (test bench)

For verification of the neural network model, a separate top module (test bench) is written in Verilog. The above figure gives an overview of the verification of a model using test data sets as input. An AXI interface between the top module and the DUT (Neural network) has been used for passing the test data set one at a time to the DUT as shown in fig. An AXI interface helps in sending input datasets in serial fashion from the top module automatically.

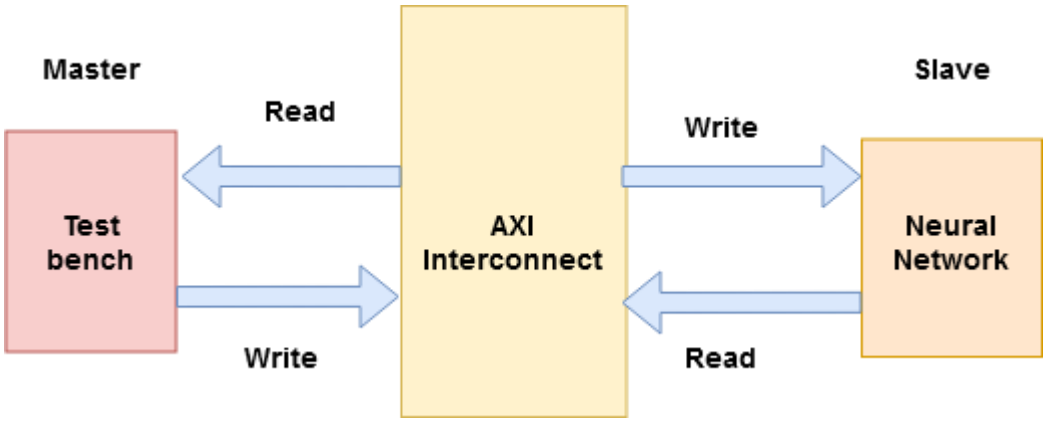

Fig 4.11: AXI interface

Chapter 5

RESULT AND DISCUSSION

In this segment of work we observe the results after behavioral simulation and post synthesis implementation of the neural network design on a suitable FPGA board. The tool employed for RTL implementation of neural network is Xilinx Vivado 2020.1 and the FPGA board targeting this work is Artix-7(xc7a100t csg324).

5.1 Simulation results:

In this design process, the design of the fundamental unit of the neural network, i.e., the neuron was the first goal. Fig 4.1 represents the behavioral simulation of the test bench given on top of the neuron module written in Verilog using the Xilinx Vivado simulator. Weighted memory and bias registers are initialized with initial contents using the MIF file added to the test bench at the start of the simulation. During simulation, each clock pulse input is driven to the module, and the weighted input computation begins. Finally, it was added to the bias value and applied to the activation ROM to produce the desired result. This simulation was performed to check the functionality of the neuron unit's operation before designing layered architecture design.

Fig 5.1: Behavioral simulation of neuron architecture

After adding layers and pipeline register in between the layers complete architecture of neural network design is simulated. Fig 4.2 represents the behavioral simulation of a fully connected neural network consisting of three hidden layers (784-30-30-10-10).

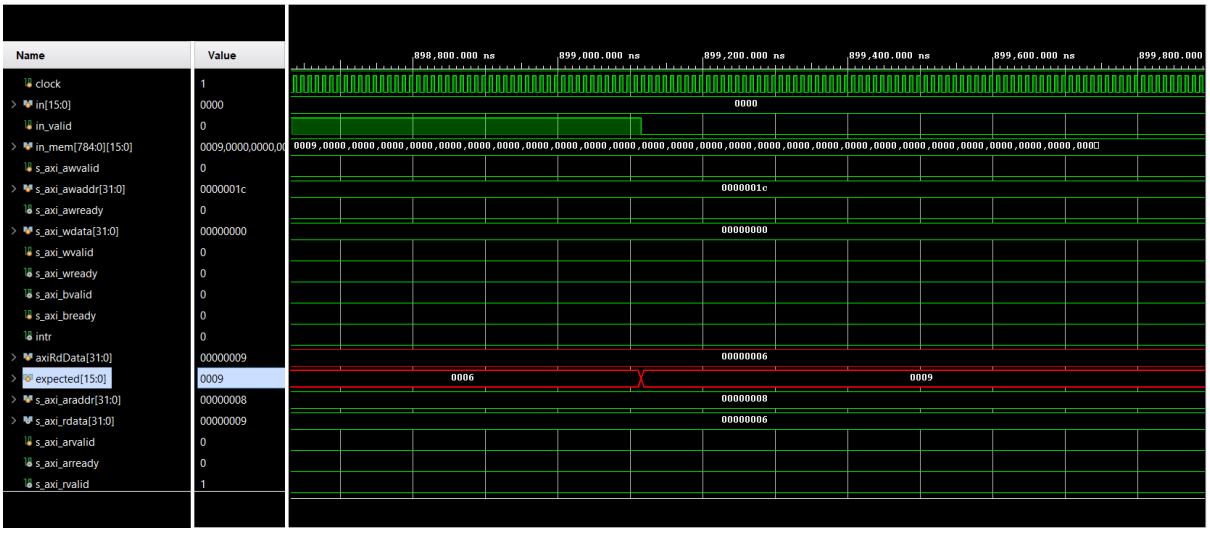

Fig 5.2: Behavioral simulation of fully connected neural network

The complete design was tested using test data sets and check against the expected one to find the accuracy of design. Fig 4.2 as shown, showing the accuracy results after 100 image data set given to neural network (784-30-30-10-10).

```
71. Accuracy: 97.183099, Detected number: 7, Expected: 0007
72. Accuracy: 97.222222, Detected number: 0, Expected: 0000
73. Accuracy: 97.260274, Detected number: 2, Expected: 0002
74. Accuracy: 97.297297, Detected number: 9, Expected: 0009
75. Accuracy: 97.333333, Detected number: 1, Expected: 0001
76. Accuracy: 97.368421, Detected number: 7, Expected: 0007
77. Accuracy: 97.402597, Detected number: 3, Expected: 0003
78. Accuracy: 97.435897, Detected number: 2, Expected: 0002
79. Accuracy: 97.468354, Detected number: 9, Expected: 0009
80. Accuracy: 97.500000, Detected number: 7, Expected: 0007
81. Accuracy: 97.530864, Detected number: 7, Expected: 0007
82. Accuracy: 97.560976, Detected number: 6, Expected: 0006
83. Accuracy: 97.590361, Detected number: 2, Expected: 0002
84. Accuracy: 97.619048, Detected number: 7, Expected: 0007
85. Accuracy: 97.647059, Detected number: 8, Expected: 0008
86. Accuracy: 97.674419, Detected number: 4, Expected: 0004
87. Accuracy: 97.701149, Detected number: 7, Expected: 0007
88. Accuracy: 97.727273, Detected number: 3, Expected: 0003
89. Accuracy: 97.752809, Detected number: 6, Expected: 0006
90. Accuracy: 97.777778, Detected number: 1, Expected: 0001
91. Accuracy: 97.802198, Detected number: 3, Expected: 0003
92. Accuracy: 97.826087, Detected number: 6, Expected: 0006
93. Accuracy: 97.849462, Detected number: 9, Expected: 0009
94. Accuracy: 97.872340, Detected number: 3, Expected: 0003
95. Accuracy: 97.894737, Detected number: 1, Expected: 0001
96. Accuracy: 97.916667, Detected number: 4, Expected: 0004
97. Accuracy: 97.938144, Detected number: 1, Expected: 0001
98. Accuracy: 97.959184, Detected number: 7, Expected: 0007
99. Accuracy: 97.979798, Detected number: 6, Expected: 0006
100. Accuracy: 98.000000, Detected number: 9, Expected: 0009
Accuracy: 98.000000
$stop called at time : 900235 ns : File "D:/kuldeep_new/major_project_fourth_sem/EXTRA/new2/verilog and python files of project/DNN3/top_sim3.v" Line 358
run: Time (s): cpu = 00:00:15 ; elapsed = 00:00:32 . Memory (MB): peak = 1013.062 ; gain = 0.000
```

Fig 5.3: Accuracy result of neural network with three hidden layers

5.2 Evaluation of model:

In this chapter, we evaluate the model on the basis of three design parameters which are:

1. Number of hidden layers.

2. Activation function type.

3. Activation function representation.

The performance parameters employed in this work are accuracy, resource utilization, and power consumption. Before going to the results evaluation part let's first define the accuracy term related to this work.

Accuracy:

By Accuracy of neural network here means how many test images it can detect correctly out of total test images. The accuracy will depend upon several factors like how many processing elements are there, the numbers of hidden layers.

$$Accuracy(\%) = \frac{Total\ numbers\ of\ test\ image\ detects\ correctly}{Total\ numbers\ of\ input\ test\ images} \times 100$$

To check the accuracy of the neural network, 100 test images were taken from the MNIST test dataset. This examination of neural networks with varied numbers of hidden layers but the same sigmoid activation function.

a) <u>Number of hidden Layers</u>:

The above table 1 describes that different number of hidden layers affects the accuracy of neural network design. It is clearly represented from above a the number of hidden layers increases the accuracy increases as more numbers of processing layers will help in predicting test image accurately. However, as we move to a network with a large number of hidden layers, the network's performance in terms of the time it takes to process those 100 test images degrades. So, if the problem's criterion is accuracy, use a large number of hidden layers; otherwise, if performance is a primary consideration, use a limited number of hidden layers.

Table 5.1: Effect of number of hidden layers on accuracy and performance

SNO.	Neural Network Models (Number of hidden layers)	Accuracy (%)	Performance(time)
1.	784 – 30 – 10 (1)	94	0.846ms
2.	784 – 30 – 30 – 10 (2)	96	0.883ms
3.	784 – 30 – 30 – 10 – 10 (3)	98	0.904ms

In this evaluation, a neural network has been analyzed with different numbers of hidden layers but the same activation function sigmoid.

Table 5.2: Effect of number of hidden layers on resource and power

Resources No. of hidden layers	LUT (Utilization %)	FF (Utilization %)	BRAM (Utilization %)	DSP (Utilization %)	IO (Utilization %)	BUFG (Utilization %)	Power (watts)
1	3256 (5.14)	2935 (2.31)	25 (18.52)	80 (33.33)	105 (50)	1 (3.13)	124.04
2	5409 (8.53)	4688 (3.77)	32.5 (24.07)	140 (58.33)	105 (50)	1 (3.13)	187.71
3	6225 (9.87)	5293 (4.17)	35 (25.93)	160 (66.67)	105 (50)	1 (3.13)	188.22

We can deduce from table 2 above that resource usage and power are directly related to the number of neurons used in neural networks.

b). <u>Type of activation function</u>:

Two types of activation functions were employed to model the neural network in this study. Both the Sigmoid and ReLU activation functions are nonlinear. So, for this evaluation, we used a neural network with three hidden layers (784-30-30-10-10).

Table 5.3: Effect of type of activation function on accuracy and resource utilization

Resources / Activation Function	LUT (Utilization %)	FF (Utilization %)	BRAM (Utilization %)	DSP (Utilization %)	IO (Utilization %)	BUFG (Utilization %)	Accuracy (%)
Sigmoid	6255 (9.87)	5293 (4.17)	35 (25.93)	160 (66.67)	105 (50)	1 (3.13)	98
ReLU	8273(13.05)	6653(5.25)	15(11.11)	160(66.67)	105(50)	1(3.13)	91

From above, it clearly states that for this neural network design, ReLU implementation consumes more resources than sigmoid. Logical utilization (LUTs and FFs) is more in ReLU than sigmoid because there is a need for extra logic in realizing ReLU. Since memory is involved in the realization of sigmoid apart from weight memory, a neural network with sigmoid consumes more block RAM (BRAM) compared to a neural network with ReLU. In terms of accuracy, sigmoid activation gives better accuracy for the same configuration as it is more nonlinearity than ReLU.

c). <u>Activation function representation</u>:

The table shows a binary representation of the activation function changing and its effect in this discussion. As we utilize more bits for representation, the model's accuracy increases but more hardware is consumed. As the number of bits in the function increases,

it covers a wider range of activation values, necessitating the use of more BRAM and LUTs for approximation.

Table 5.4: Effect of activation function representation on accuracy and resource utilization

Resources / Activation Function	LUT (Utilization %)	FF (Utilization %)	BRAM (Utilization %)	DSP (Utilization %)	IO (Utilization %)	BUFG (Utilization %)	Accuracy (%)
6 bits	6240 (9.84)	5293 (4.17)	35 (25.93)	160 (66.67)	105 (50)	1 (3.13)	94.6
12 bits	16227(25.59)	5879 (4.64)	98 (72.59)	160(66.67)	105(50)	1(3.13)	95.4

Chapter 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

An FPGA-based fully connected neural network is designed in this thesis. Since FPGA has a reconfigurable and parallel architecture, it is a great platform for deep neural networks. The reconfigurability of FPGA allows neural network architectures to be scalable and flexible to change as they are used in this thesis. When examined closely, a neural network is nothing more than a large number of neurons arranged in layered architecture doing the same arithmetic computation. As each neuron does the same repeated action individually, the FPGA's parallel computing architecture can be used to conduct independent operations concurrently. The fully connected neural network is designed to address problems related to the recognition of handwritten digits using MNIST datasets. The design is implemented on an Artix-7 FPGA board with varying design parameters like number of layers, activation functions etc. The FPGA-based design of neural networks utilizes pre-trained weights and biases. The design performance was carried out in terms of accuracy and FPGA resource utilization. The design attained a maximum accuracy of 98 % by altering its design parameters. This architecture can be employed when a machine learning problem has a minimal number of data sets, few calculations, and a straightforward design.

6.2 Future Scope

Although the fully connected neural network model correctly predicts the images in this study, there are still many ways to improve the accuracy and efficiency of this design. There are three key areas where this design will need to improve in the future to maximize its performance. The first is neural network architecture, which is preferable to fully connected architecture since it is more difficult to construct but more computationally efficient, and so performs better when it comes to image recognition.

The representation of data is the second area in which design should be concentrated. Fixed-point representation was used in this design because of its simplicity and low resource usage. If accuracy is more important than resource utilization, floating-point data representation is a great option. There are two major advantages to using floating-point. One is that this type of representation includes a scaling factor that allows for a greater range of values to be represented. They're also more adaptable than fixed-point models.

The third and most essential area needs to concentrate on using FPGAs to approximate activation functions. There are numerous efficient approaches that can be used in this design. There are a few good approximation approaches used in FPGA implementation, such as piecewise approximation and LUT-based approximation.

REFERENCES

[1] P. P. Shinde and S. Shah, "A Review of Machine Learning and Deep Learning Applications," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018.

[2] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *nature*, *521*(7553), pp.436-444.

[3] Sze, V., Chen, Y.H., Yang, T.J. and Emer, J.S., 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, *105*(12), pp.2295-2329.

[4] Ersoy, Mevlüt & Kumral, Cem. (2020). Realization of Artificial Neural Networks on FPGA. 10.1007/978-3-030-36178-5_31.

[5] O. I. Abiodun et al., "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition," in IEEE Access, vol. 7, pp. 158820-158846, 2019, doi: 10.1109/ACCESS.2019.2945545.

[6] Gu, Y., Shibukawa, T., Kondo, Y., Nagao, S. and Kamijo, S., 2020. Prediction of stock performance using deep neural networks. *Applied Sciences*, *10*(22), p.8142.

[7] M. Ananda Ro and J. Srinivas, Neural Networks Algorithms and Applications, Alpha Science International Ltd., Part Ill, pp.157, 2003.

[8] P. Ongsulee, "Artificial intelligence, machine learning and deep learning," *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*, 2017, pp. 1-6, doi: 10.1109/ICTKE.2017.8259629.

[9] Ongsulee, P., 2017, November. Artificial intelligence, machine learning and deep learning. In *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)* (pp. 1-6). IEEE.

[10] T. V. Huynh, "Deep neural network accelerator based on FPGA," *2017 4th NAFOSTED Conference on Information and Computer Science*, 2017, pp. 254-257, doi: 10.1109/NAFOSTED.2017.8108073.

[11] C. A. R. de Sousa, "An overview on weight initialization methods for feedforward neural networks," *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 52-59, doi: 10.1109/IJCNN.2016.7727180.

[12] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[13] Baldominos, A., Saez, Y. and Isasi, P., 2019. A survey of handwritten character recognition with mnist and emnist. *Applied Sciences*, *9*(15), p.3169.

[14] Parth Bhasin, Vaishali, 2017, Back Propogation Algorithm: An Artificial Neural Network Approach, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ICCCS – 2017.

[15] E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala and C. O. Aigbavboa, "A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks," 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), 2018, pp. 92-99, doi: 10.1109/CTEMS.2018.8769211.

[16] Sánchez, Alberto & De Castro, Angel & Martínez-García, María & Garrido, Javier. (2020). LOCOFloat: A low-cost floating-point format for FPGAs.: Application to HIL simulators. Electronics. 9. 81. 10.3390/electronics9010081.

[17] J. J. Rodríguez-Andina, M. D. Valdés-Peña and M. J. Moure, "Advanced Features and Industrial Applications of FPGAs—A Review," in *IEEE Transactions on Industrial Informatics*, vol. 11, no. 4, pp. 853-864.

[18] K. Huang, M. Gungor, S. Ioannidis and M. Leeser, "Optimizing Use of Different Types of Memory for FPGAs in High Performance Computing," *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1-7.