1. INTRODUCTION

1.1 INTRODUCTION TO NEURAL NETWORKS

The human nervous system has been treated by numerous studies. Not only from a medical point of view, as treatments for prevention of diseases, but also, from a technological point of view, trying to emulate the behavior of the biological neuron. And based on this, modelling an artificial neuron can be used to develop future applications in computational neuroscience, as well as in artificial intelligence.

Artificial neural networks simplify the behavior of the human brain, so, their applications are used in different fields as industrial automation, medicinal applications, robotics, electronics, security, transport, military, etc.

Applications in pattern recognition like recognition of fingerprints or control of missiles use systems based on neural networks among other techniques. The following research is based exactly in this issue, to understand the real behaviour of a biological neuron and according to this being able to model an artificial neuron that works in a similar way. When this artificial neuron will be developed, it will be used in future applications through complete neural networks for applications as commented previously.

A neuron can be compared to a black box composed of few inputs and an output. Like an electrical circuit that makes the addition of the different signals that receives from other units and obtain in the output according to the result of the addition with relation to the threshold. The artificial neuron is an electronic device that responds to electrical signals, an example of the use of artificial neurons in the industry is, for example, CCortex, building by CorticalDB, which is a massive spiking neural network simulation of the human cortex and peripheral systems. CCortex represents up to 20 billion neurons and 20 trillion connections.

According to the manufacturer, the development of CCortex will enable a wide range of commercial products that will transform and enhance global business relationships, with advanced capabilities in pattern recognition, verbal and visual communication, knowledge acquisition, and decision-making capabilities. These products will have widespread applications in the fields of artificial intelligence, communications, medical modeling, and database and search technologies.

These are only a little sample of the abilities that can be implemented starting with the development of the artificial neuron.

1.2 BIOLOGICAL BACKGROUND

First at all, we must study and understand the different parts and the behavior of neurons. Functionally, a neuron can be divided in three parts as in Figure 1.1:

Soma: The body of the neuron.

Dendrites: Branches that transmit the action potentials from others neurons to the soma.

Axon: A branch that transmits the action potential from the neuron to other cell.

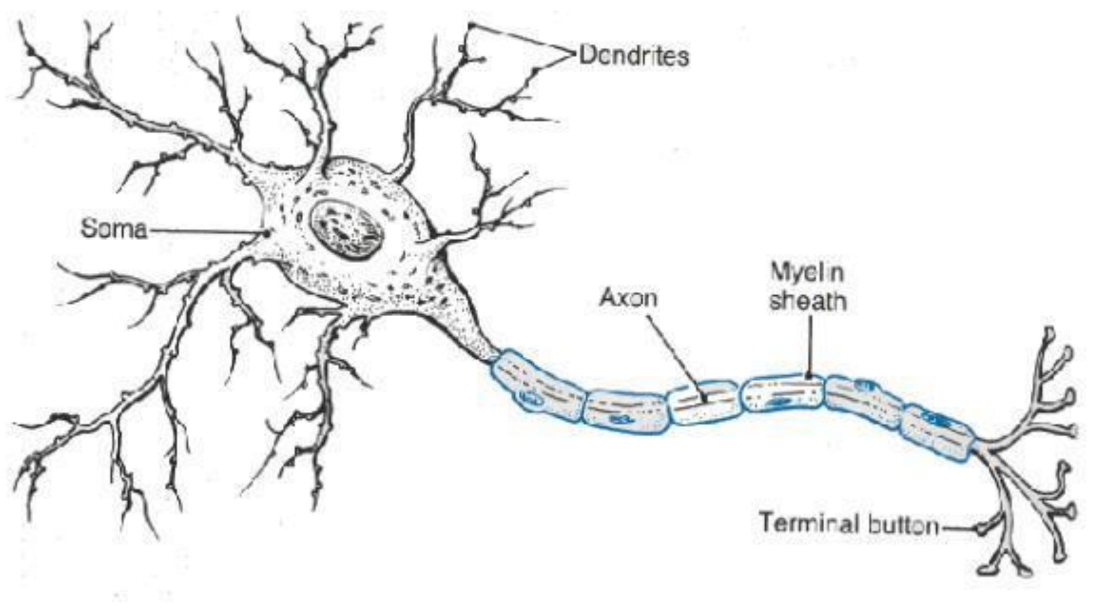


Figure 1.1: Parts of a neuron.

The soma is the central unit of the neuron, it receives signals from others neurons across the different dendrites. If the addition of these signals is over of a certain threshold an output signal, a spike, is generated. The spike is propagated by the axon to other neurons. The contact site between the axon of the first neuron and the dendrites of the next neurons is called synapse.

The spikes are short electrical pulses. The amplitude of theses spikes, are about 100 mV and they have duration of 1 or 2 ms.

A spike train is a chain of action potentials emitted by a neuron, the importance of action potentials is in the timing and the numbers of spikes, the form of the action potential does not carry any information, since they all look almost the same. The spike is the elementary unit of signal transmission.

When a neuron is fired, it generates an action potential, this action potential is transmitted and processed in the soma of the next neuron, where action potential from others neurons are arriving. The soma considers all the input from the different neurons and makes a non-linear addition of these signals. The sum of this addition gives the membrane potential of the neuron, which is transmitted along the axon to target neurons.

During the summation process, not all the inputs have the same relevance, thus, some are more important (represented as higher weight values) whereas other inputs are less important (lower weight values). One of the characteristics of the biological as well as artificial neuronal network is their capacity of learning, for that, the inputs have adaptive weights.

1.3 HOPFIELD

The network we have encountered so far are known as feed forward networks since they have no loops in them. In such networks, the relationship between the input and the output can be expressed as a static function. But loops are more common in the brain.

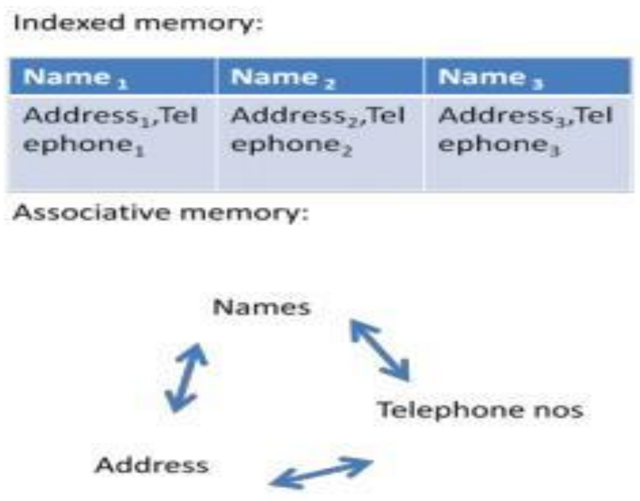
1) A cortical area A that projects to another cortical area B, very often receives a feedback projection from B, forming a loop. Such connections are called reentrant connections.

2) Or a cortical area can project in a series of sub-cortical nuclei, ultimately received feedback from a nucleus at the end of a long chain, forming a loop.
3) Neurons in cortex and deep brain nuclei typically have local, lateral connections among themselves forming loops at microscopic level.

Thus loops are more common in the brain than cascades of neural layers connected in Feed forward fashion.

Hopfield network is a simple neural network model that has feedback connections. It significance lies in the fact that it was able to bring together ideas from neurobiology and psychology and present a model of human memory, known as an associative memory.

The concept of an associative memory can be best explained by contrasting it with computer memory, which is known as indexed memory.



Name $_1$	Name $_2$	Name $_3$
Address$_1$,Telephone$_1$	Address$_2$,Telephone$_2$	Address$_3$,Telephone$_3$

Figure 1.2: Indexed memory and Associative memory

Indexed memory consists of two columns of data. The first column refers to addresses and the second to contents. If you know the address you can fetch the content located at that address.

A simple example of an indexed memory from the real world is a traditional telephone directory. The “addresses” are the names of customers. Contents are a combination of physical address, telephone number and other details of the customers.

Such a directory has certain disadvantages. It is not possible to fetch the name with the help of a telephone number, or fetch address using telephone number.

But in an associative memory there is no separate “address” and “content.” We store memories as networks of items. Each item acts as a cue to retrieve other items, which in turn can cue other items. A given item can act as an “address” or “content” depending on the context.

A telephone directory organized as an associative memory becomes immediately more usable.

Instead of splitting a record into “address” (customer’s name) and “content” (rest of the record), we can consider splitting it into three fields.
Customer’s name<-> physical address<->telephone number

Any of these fields can uniquely identify the remaining two. Storage and retrieval from such a memory must also be conducted on very different lines from an indexed memory. The Hopfield network is essentially a neural network realization of an associative memory. It is possible to have non-neural network-based based associative memories. See Kanerva’s
memories for example. But they are not the relevant for the present discussion.

1.4 COGNITIVE SCIENCE

Cognitive science can be roughly summed up as the scientific interdisciplinary study of the mind. Its primary methodology is the scientific method, although as we will see, many other methodologies also contribute. A hallmark of cognitive science is its interdisciplinary approach. It results from the efforts of researchers working in a wide array of fields. These include philosophy, psychology, linguistics, artificial intelligence, robotics, and neuroscience. Each field brings with it a unique set of tools and perspectives. One major goal of this book is to show that when it comes to studying something as complex as the mind, no single perspective is adequate.

Instead, Inter-communication and cooperation among the practitioners of these disciplines tell us much more.

The term cognitive science refers not so much to the sum of all these disciplines but to their intersection or converging work on specific problems. In this sense, cognitive science is not a unified field of study like each of the disciplines themselves, but a collaborative effort among researchers working in the various fields. The glue that holds cognitive science together is the topic of mind and, for the most part, the use of scientific methods. In the concluding chapter, we talk more about the issue of how unified cognitive science really is
In order to really understand what cognitive science is all about we need to know what its theoretical perspective on the mind is. This perspective centers on the idea of computation, which may alternatively be called information processing. Cognitive scientists view the mind as an information processor. Information processors must both represent and transform information. That is, a mind, according to this perspective, must incorporate some form of mental representation and processes that act on and manipulate that information. We will discuss these two ideas in greater detail later in this chapter.

Cognitive science is often credited with being influenced by the rise of the computer. Computers are of course information processors. Think for a minute about a personal computer. It performs a variety of information processing tasks. Information gets into the computer via input devices, such as a keyboard or modem. That information can then be stored on the computer, for example, on a hard drive or other disk. The information can then be processed using software such as a text editor. The results of this processing may next serve as output, either to a monitor or printer. In like fashion, we may think of people performing similar tasks. Information is "input" into our minds through perception—what we see or hear. It is stored in our memories and processed in the form of thought. Our thoughts can then serve as the basis of "outputs," such as language or physical behavior.

Of course this analogy between the human mind and computers is at a very high level of abstraction. The actual physical way in which data is stored on a computer bears little resemblance to human memory formation. But both systems are characterized by computation.

In fact, it is not going too far to say that cognitive scientists view the mind as a machine or mechanism whose workings they are trying to understand.

As the title suggests our project deals with a hardware implementation of artificial neural networks, specifically a FPGA implementation. During the course of this project we learnt about ANNs and the uses of such soft computing approaches, FPGAs, VHDL and use of various tools like Xilinx ISE Project Navigator and ModelSim. As numerous hardware implementations of ANNs already exist our aim was to come up with an approach that would facilitate topology evolution of the ANN as well.

The key problem in the simulation of ANN's is its computational overhead. Networks containing millions of neurons and ten billion connections, and complex models like spiking neurons with temporal time course that require convolutions to be computed at each synapse, will challenge even the fastest computers. Hence there is much interest in developing custom hardware for ANN's. Points in favor of hardware implementations are:

a) Inherent parallelism and connectionist model of ANN's which find a natural application through hardware. General purpose processors operate sequentially.
b) Simple ANN models require simple, low precision computations which can be performed faster on cheap and low precision hardware. Also since hardware is getting cheaper by the day, custom hardware can be built to perform complex computations.

Field Programmable Gate Arrays (FPGA) are a type of hardware logic device that have the exibility to be programmed like a general-purpose computing platform (e.g. CPU), yet retain execution speeds closer to that of dedicated hardware (e.g. ASICs). Traditionally, FPGAs have been used to prototype Application Specific Integrated Circuits (ASICs) with the intent of being replaced in _nal production by their corresponding ASIC designs. Only in the last decade have lower FPGA prices and higher logic capacities led to their application beyond the prototyping stage, in an approach known as reconfigurable computing. A question remains concerning the degree to which reconfigurable computing has benefited from recent improvements in the state of FPGA technologies / tools. This thesis presents a Reconfigurable Architecture for Implementing ANNs on FPGAs as a case study used to answer this question.

1.5 EXISTING SYSTEM

Back Propagation is Solution Network, because of the nature of the activation function, the activity on the output node can never reach either ‘0’ or ‘1’. We take values of less than 0.1 as equal to 0, and greater than 0.9 as equal to 1.If the network seems to be stuck, it has hit what is called a ‘local minimum’. Keep your eye on the bias of the hidden node and wait. It will eventually head towards zero. As it approaches zero, the network will get out of the local minimum, and will shortly complete. This is because of a ‘momentum turn’ that is used in the calculation of the weights.

Conditional Back propagation Network:

This network can learn any logical relationship expressible in a truth table of this sort. In the following, you can change the desired output, and train the network to produce that output.

Back propagation for Any Binary Logical Function:

This network makes use of binary values and is used in less iterative steps. It’s most handy and is quicker in getting the solution.

MLP (Multi Layered Perceptron):

This is a complex network with more basic nodes used and here, the feature value is set.

1.6 PROPOSED SYSTEM

The XOR, or “exclusive or”, problem is a classic problem in ANN research. It is the problem of using a neural network to predict the outputs of XOR logic gates given two binary inputs. An XOR function should return a true value if the two inputs are not equal and a false value if they are equal.

XOR is a classification problem and one for which the expected outputs are known in advance. It is therefore appropriate to use a supervised learning approach.

On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s, known as perceptrons.

Perceptrons: Like all ANNs, the perceptron is composed of a network of units, which are analogous to biological neurons. A unit can receive an input from other units. On doing so, it takes the sum of all values received and decides whether it is going to forward a signal on to other units to which it is connected. This is called activation. The activation function uses some means or other to reduce the sum of input values to a 1 or a 0 (or a value very close to a 1 or 0) in order to represent activation or lack thereof. Another form of unit, known as a bias unit, always activates, typically sending a hard coded 1 to all units to which it is connected.

Perceptrons include a single layer of input units — including one bias unit — and a single output unit (see figure 1.2). Here a bias unit is depicted by a dashed circle, while other units are shown as blue circles. There are two non-bias input units representing the two binary input values for XOR. Any number of input units can be included.

The perceptron is a type of feed-forward network, which means the process of generating an output

— known as forward propagation — flows in one direction from the input layer to the output layer. There are no connections between units in the input layer. Instead, all units in the input layer are connected directly to the output unit.

A simplified explanation of the forward propagation process is that the input values X1 and X2, along with the bias value of 1, are multiplied by their respective weights W0...W2, and parsed to the output unit. The output unit takes the sum of those values and employs an activation function

— Typically the Heaviside step function — to convert the resulting value to a 0 or 1, thus classifying the input values as 0 or 1.

It is the setting of the weight variables that gives the network's author control over the process of converting input values to an output value. It is the weights that determine where the classification line, the line that separates data points into classification groups, is drawn. If all data points on one side of a classification line are assigned the class of 0, all others are classified as 1.

Multilayer Perceptrons: The solution to this problem is to expand beyond the single-layer architecture by adding an additional layer of units without any direct access to the outside world, known as a hidden layer. This kind of architecture — shown in Figure 4 — is another feed-forward network known as a multilayer perceptron (MLP).

It is worth noting that an MLP can have any number of units in its input, hidden and output layers. There can also be any number of hidden layers. The architecture used here is designed specifically for the XOR problem.

Similar to the classic perceptron, forward propagation begins with the input values and bias unit from the input layer being multiplied by their respective weights, however, in this case there is a weight for each combination of input (including the input layer's bias unit) and hidden unit (excluding the hidden layer's bias unit). The products of the input layer values and their respective weights are parsed as input to the non-bias units in the hidden layer. Each non-bias hidden unit invokes an activation function — usually the classic sigmoid function in the case of the XOR problem — to squash the sum of their input values down to a value that falls between 0 and 1 (usually a value very close to either 0 or 1). The outputs of each hidden layer unit, including the bias unit, are then multiplied by another set of respective weights and parsed to an output unit. The output unit also parses the sum of its input values through an activation function — again, the sigmoid function is appropriate here — to return an output value falling between 0 and 1. This is the predicted output.

This architecture, while more complex than that of the classic perceptron network, is capable of achieving non-linear separation. Thus, with the right set of weight values, it can provide the necessary separation to accurately classify the XOR inputs.

Back propagation

The elephant in the room, of course, is how one might come up with a set of weight values that ensure the network produces the expected output. In practice, trying to find an acceptable set of weights for an MLP network manually would be an incredibly laborious task. In fact, it is NP-complete (Blum and Rivest, 1992). However, it is fortunately possible to learn a good set of weight values automatically through a process known as back propagation. This was first demonstrated to work well for the XOR problem by Rumelhart et al. (1985).

The back propagation algorithm begins by comparing the actual value output by the forward propagation process to the expected value and then moves backward through the network, slightly adjusting each of the weights in a direction that reduces the size of the error by a small degree. Both forward and back propagation are re-run thousands of times on each input combination until the network can accurately predict the expected output of the possible inputs using forward propagation.

For the XOR problem, 100% of possible data examples are available to use in the training process. We can therefore expect the trained network to be 100% accurate in its predictions and there is no need to be concerned with issues such as bias and variance in the resulting model.

1.7 APPLICATIONS

Future Electronics has a wide range of programmable logic gates from several chip manufacturers that can be used for programming your circuits. By going through the selection you will find the right digital logic gate, AND gate, OR gate, CMOS logic gate, NOT gate, XOR gate, NOR gate, NAND gate, universal logic gate, electronic logic gate, transistor logic gate or chips for any type of circuit or IC that might require logic gates. Once you decide if you need a dual gate, quad gate, single gate of triple gate, you will be able to choose from their technical attributes and your search results will be narrowed to match your specific logic gate application needs.

We deal with several manufacturers such as Fairchild, Diodes Inc., NXP, ON Semiconductor or STMicroelectronics, among others. You can easily refine your logic gate product search results by clicking your preferred logic gate brand from the list of manufacturers below.

Applications for Logic Gates:

Logic circuits are found in several devices including multiplexers, arithmetic logic units, computer memory and registers. They are also used in microprocessors, some of which can contain over 100 million gates. Also, logic gates are the building blocks of digital electronics and are formed by the combination of transistors in order to realize some digital operations. Every digital product, including personal computers, mobile phones, tablets, calculators and digital watches also uses logic gates.

1.8 REPORT ORGANIZATION

In this report we first give the introduction of the XOR problem and FPGA's role in hardware implementation of neural network. Next we give the software and hardware requirements for the project. We then give the design of the project in the form. Next we give the implementation of the project. We then proceed to the testing of the project and give the results. We finally proceed to conclude the project report.

Chapter 1 deals with the Introduction of this project.

Chapter 2 deals with the Literature Survey.

Chapter 3 deals about System Requirements and

Analysis.

Chapter 4 deals with Design of the project.

Chapter 5 deals with the Implementation part.

Chapter 6 deals with Testing and Results.

Chapter 7 deals with the Conclusion and Future scope.

2. LITERATURE SURVEY

2.1 COMPARISON BETWEEN ANALOG AND DIGITAL IMPLEMENTATIONS OF ANN'S

Intuitively analog seems a better choice as it would be better to represent synaptic weights by analog quantities. Computational density of analog chips is greater. Complex, non-linear functions like multiply, divide and hyperbolic tangent can be performed with a handful of transistors. Power required for these computations is less than with digital methods.

However, people are more familiar with digital design now i.e. analog design is an uncommon capability. Most applications comprise of digital systems and a digital neural network will provide ease in integration with these systems.

Analog designs are hardwired and thus inflexible. Digital designs are flexible as in they employ part software control, arbitrary precision and re-programmability. This enables them to solve a larger part of the problem at the price of reduced performance/cost.
Signal interconnection is also a problem. Wire interconnections in silicon are expensive and take up more area. The solution has to be multiplexing of connections i.e. multiple synaptic connections share the same wire. This adds to the complexity if the design but results in a massive reduction in cost. Analog designs are hard to multiplex.
Amadahl's law states that no matter how many processors are available to execute a subtask, the speed of a particular task is roughly proportional to the number of subtasks that have to be executed sequentially. Thus a programmable device that accelerates several phases of an application offers more benefits than a dedicated device.

2.2 FPGA VS OTHER CHIPS

There are several problems here, not the least that the technologies and terminologies have evolved over time. Keeping this in mind, the following is my highly simplified interpretation of where these terms came from and what they mean today

2.2.1 ASICs

Let's start with an application-specific integrated circuit (ASIC). As the name suggests, this is a device that is created with a specific purpose in mind. When most people hear the term ASIC, their "knee-jerk" reaction is to assume a digital device. In reality, any chip that is custom-made is an ASIC, irrespective of whether it is analog, digital, or a mix of both. For the purposes of these discussions, however, we shall assume a chip that is either wholly or predominantly digital in nature, with any analog and mixed-signal functions being along the lines of physical interfaces (PHYs) or phase-locked loops (PLLs).

ASICs are typically designed and used by a single company in a specific system. They are incredibly expensive, time-consuming, and resource-intensive to develop, but they do offer extremely high performance coupled with low power consumption.

2.2.2 ASSPs

Application-specific standard parts (ASSPs) are designed and implemented in exactly the same way as ASICs. This is not surprising, because they are essentially the same thing. The only difference is that an ASSP is a more general-purpose device that is intended for use by multiple system design houses. For example, a standalone USB interface chip would be classed as an ASSP.

2.2.3 SoCs

A System-on-Chip (SoC) is a silicon chip that contains one or more processor cores -- microprocessors (MPUs) and/or microcontrollers (MCUs) and/or digital signal processors (DSPs) -- along with on-chip memory, hardware accelerator functions, peripheral functions, and (potentially) all sorts of other "stuff." One way to look at this is that if an ASIC contains one or more processor cores then it's a SoC. Similarly, if an ASSP contains one or more processor cores then it's a SoC.

On this basis, we could view ASIC (and ASSP) as being the superset term because it embraces SoC, or we could regard the SoC as being the superset term because it includes everything in an ASIC (or ASSP) along with one or more processor cores.

2.2.4 FPGA’s

ASICs, ASSPs, and SoCs offer high-performance and low power consumption, but any algorithms they contain -- apart from those that are executed in software on internal processor cores -- are “frozen in silicon.” And so we come to field-programmable gate arrays (FPGAs). The architecture of early FPGA devices was relatively simple -- just an array of programmable blocks linked by programmable interconnect.

The great thing about an FPGA is that we can configure its programmable fabric to implement any combination of digital functions we desire. Also, we can implement algorithms in a massively parallel fashion, which means we can perform a humongous amount of data processing very quickly and efficiently.

2.2.5 SoC-CLASS FPGA’s

Over time, the capabilities (capacity and performance) of FPGAs increased dramatically. For example, a modern FPGA might contain thousands of adders, multipliers, and digital signal processing (DSP) functions; megabits of on-chip memory, large numbers of high-speed serial interconnect (SERDES) transceiver blocks, and a host of other functions.

The problem is that the field-programmable gate array (FPGA) moniker no longer reflects the capabilities and functionality of today's programmable devices. We really need to come up with some new terminology that embraces everything today's state-of-the-art tools and technologies are capable of doing.

Of particular relevance to our discussions here is the fact that today's FPGAs can contain one or more soft and/or hard core processors. On this basis, should we class this type of FPGA as being a SoC? Well, personally I have to say that SoC doesn’t work for me, because I equate the term "SoC" with a custom device created using ASIC technology.

Another alternative would be to call these devices Programmable SoCs, or PSoCs, but Cypress Semiconductor has already got the PSoC moniker locked down. The Cypress devices feature a hard microcontroller core augmented with some programmable analog and programmable digital fabric (the digital fabric is more CPLD than FPGA).

Altera used to call its versions of these devices -- the ones that combine hard MCU cores with programmable FPGA fabric -- SoC FPGAs, but they seem to have evolved to just calling them SoCs. Meanwhile, Xilinx calls its flavor of these devices "All Programmable SoCs."

2.3 OVERVIEW OF VHDL

VHDL is a language meant for describing digital electronic systems. In its simplest form, the description of a component in VHDL consists of an interface specification and an architectural specification. The interface description begins with the ENTITY keyword and contains the input-output ports of the component. The name of the component comes after the ENTITY keyword and is followed by IS, which is also a VHDL keyword. The description of the internal implementation of an entity is called an architecture body of the entity. There may be a number of different architecture bodies of an interface to an entity corresponding to alternative implementations that perform the same function. The alternative implementations of the architecture body of the entity are termed as Behavioral Description or Structural Description.

After describing a digital system in VHDL, simulation of the VHDL code has to be carried out for two reasons. First, we need to verify whether the VHDL code correctly implements the intended design. Second, we need to verify that the design meets its specifications. The simulation is used to test the VHDL code by writing test bench models. A test bench is a model that is employed to exercise and verify the correctness of a hardware model and it can be described in the same language. Some synthesis tools are capable of implementing the digital system described by the VHDL code using a PGA (Programmable gate array) or CPLD (Complex programmable logic devices). The PLDs are capable of implementing a sequential network but not a complete digital system. Programmable gate arrays and complex programmable logic devices are more flexible and more versatile and can be used to implement a complete digital system on a single chip. A typical FPGA is an IC that contains an array of identical logic cells with programmable interconnections. The user can program the functions realized by each logic cell and the connections between the cells. Such PGAs are often called FPGAs since they are field programmable.

2.4 HARDWARE IMPLEMENTATIONS USING FPGAS

This is the most promising method. The structure of FPGA's is suitable for implementations of ANN's.

FPGA-based reconfigurable computing architectures are well suited for implementations of neural networks as one may exploit concurrency and rapidly reconfigure for weight and topology adaptation. All FPGA implementations attempt to exploit the re-configurability if FPGA's.

Identifying the purpose of reconfiguration sheds light on the different implementation approaches.

1. **Prototyping and simulation**: FPGA's can be reconfigured an innumerable number of times. This allows rapid prototyping and direct simulation in hardware, which also increases the speed of the simulation and can be used to test various ANN implementations and learning algorithms.
2. **Density enhancement:** This is the increase in the functional capacity per unit area of the chip. Density enhancement can be done in two ways.
   A) It is possible to time multiplex an FPGA chip for each of the sequential steps in an ANN algorithm. Each stage occupies the same entire hardware resource and the resource is configured for the stage which is reached.
   B) Dynamic constant folding: In this case the FPGA chip is time multiplexed for each of the ANN circuits that is specialized with a step of constant operands in different stages during execution.

   Both these methods incur a reconfiguration time overhead and are suitable only until a certain breakeven point below which the computation time is much smaller than reconfiguration time.
3. **Topology adaptation:** Dynamically reconfigurable FPGA's permit ANN algorithm with topology adaptation along with weight adaptation.

The role which a FPGA-based platform plays in neural network implementation, and what part(s) of the algorithm it's responsible for carrying out, can be classified into two styles of architecture, as either a co-processor or as a stand-alone architecture. When taking on the role of a co-processor, a FPGA-based platform is dedicated to offloading computationally intensive tasks from a host computer. In other words, the main program is executed on a general-purpose computing platform, and certain tasks are assigned to the FPGA-based coprocessor to accelerate their execution. For neural networks algorithms in particular, an FPGA-based co-processor has been traditionally used to accelerate the processing elements (eg. Neurons).

On the other hand, when a FPGA-based platform takes on the role of a stand-alone architecture, it becomes self-contained and does not depend on any other devices to function. In relation to a co-processor, a stand-alone architecture does not depend on a host computer, and is responsible for carrying out all the tasks of a given algorithm.

3. REQUIREMENTS, ANALYSIS AND SPECIFICATIONS

3.1 INTRODUCTION

Xilinx ISE(Integrated Synthesis Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors.[3] The Xilinx ISE is primarily used for circuit synthesis and design, while ISIM or the ModelSim logic simulator is used for system-level testing.[4][5] Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and ChipScope Pro.[6]

Since 2012, Xilinx ISE has been discontinued in favor of Vivado Design Suite, that serves the same roles as ISE with additional features for system on a chip development.[7] Xilinx released the last version of ISE in October 2013 (version 14.7), and states that "ISE has moved into the sustaining phase of its product life cycle, and there are no more planned ISE releases.

The ISE Design Suite: Embedded Edition includes Xilinx Platform Studio (XPS), Software Development Kit (SDK), large repository of plug and play IP including MicroBlaze™ Soft Processor and peripherals, and a complete RTL to bit stream design flow. Embedded Edition provides the fundamental tools, technologies and familiar design flow to achieve optimal design results. These include intelligent clock gating for dynamic power reduction, team design for multi-site design teams, design preservation for timing repeatability, and a partial reconfiguration option for greater system flexibility, size, power, and cost reduction.SPA Toolbox

To address the need for performance analysis and benchmarking techniques, the Xilinx Software Development Kit (SDK) has been enhanced with a System Performance Analysis (SPA) toolbox to provide early exploration of hardware and software systems. Specifically, a Zynq-7000 AP SoC designer is presented with insights into both the PS and the PL to understand the interactions across such a complex, heterogeneous system. You can observe system performance at critical stages of a design flow, enabling you to refine the performance of your system.

3.2 REQUIREMENTS

The requirements include the following:

Operating System Support: Xilinx officially supports Microsoft Windows, Red Hat Enterprise 4, 5, & 6 Workstations (32&64 bits) and SUSE Linux Enterprise 11 (32&64 bits).Certain other GNU/Linux distributions can run Xilinx ISE WebPack with some modifications or configurations, including Gentoo Linux, Arch Linux, FreeBSD and Fedora.

3.2.1 DEVICE SUPPORT

	ISE Webpack (free)	ISE Design Suite (commercial)
VirtexFPGA	Virtex-4 LX: XC4VLX15, XC4VLX25 SX: XC4VSX25 FX: XC4VFX12 Virtex-5 LX: XC5VLX30, XC5VLX50 LXT: XC5VLX20T - XC5VLX50T FXT: XC5VFX30T Virtex-6 XC6VLX75T [10]	Virtex-4 LX: All SX: All FX: All Virtex-5 LX: All LXT: All SXT: All FXT: All Virtex-6 All
Spartan FPGA	Spartan-3 XC3S50 - XC3S1500 Spartan-3A All Spartan-3AN All Spartan-3A DSP XC3SD1800A Spartan-3E All Spartan-6	Spartan-3 All Spartan-3A All Spartan-3AN All Spartan-3 DSP All Spartan-3E All Spartan-6

	XC6SLX4 - XC6SLX75T XA (Xilinx Automotive) Spartan-6 All	All XA (Xilinx Automotive)
CoolrunnerPLA Coolrunner-II CPLD Coolrunner-IIA CPLD	All	
XC9500 Series CPLD	All (Except 9500XV family)	

3.3 ANALYSIS

3.3.1 SYSTEM PERFORMANCE

Xilinx® Software Development Kit (SDK) is delivered with a predefined project that enables System Performance Modeling (SPM) and helps enable performance analysis at an early design stage. The SPM project contains both software executable and a post-bit stream, configurable hardware system. The SPM serves multiple purposes, including the following:

• Target Evaluation – Gain a better understanding of the target platform with minimal hardware design knowledge or experience. A complex SoC such as the Zynq®-7000 AP SoC can be characterized and evaluated, enabling critical partitioning trade-offs between the ARM Cortex A9s and the programmable logic. Most importantly, this evaluation can be done independently of the progression or completion of a design.You can always decide to gain a better understanding of your target platform.

• Perform Stress Tests – Identify and exercise the various points of contention in the system. An excellent way of understanding a target platform is to evaluate the limits of its capabilities.

• Performance Evaluation – Model the traffic of a design. Once the target platform is understood, specifics of a design can be entered as traffic scenarios, and SPM can be used to evaluate architectural options.

• Performance Validation – Instrument the final design and confirm results with the initial model. The same monitoring used in the SPM design can also be added to your design (see Instrumenting Hardware, page 61), providing a validation of the modeled performance results.

3.2.2 MONITOR FRAMEWORK

The performance analysis toolbox in the Xilinx® Software Development Kit (SDK)offers aset of system-level performance measurements. For a design that targets the Zynq®7000AP SoC, this includes performance metrics from both the Programmable Logic (PL) and the Processing System (PS).

The PL performance metrics include the following:

- (Write/Read) Transactions – number of AXI transactions
- (Write/Read) Throughput – write or read bandwidth in MB/sec
- Average (Write/Read) Latency – average write or read latency of AXI transactions

The PS performance metrics include the following:

- CPU Utilization (%) – percentage of non-idling CPU clock cycles
- CPU Instructions Per Cycle (IPC) – estimated number of executed instructions per cycle
- L1 Data Cache Access and Miss Rate (%) – number of L1 data cache accesses and the miss rate
- CPU (Write/Read) Stall Cycles Per Instruction - estimated number of stall cycles per instruction due to memory writes (write) and data cache refills (read)

3.4 USER INTERFACE

The primary user interface of the ISE is the Project Navigator, which includes the design hierarchy (Sources), a source code editor (Workplace), an output console (Transcript), and a processes tree (Processes).

The Design hierarchy consists of design files (modules), whose dependencies are interpreted by the ISE and displayed as a tree structure.[3] For single-chip designs there may be one main module, with other modules included by the main module, similar to the main() subroutine in C++ programs. Design constraints are specified in modules, which include pin configuration and mapping.

The Processes hierarchy describes the operations that the ISE will perform on the currently active module. The hierarchy includes compilation functions, their dependency functions, and other utilities. The window also denotes issues or errors that arise with each

function.
The Transcript window provides status of currently running operations, and informs engineers on design issues. Such issues may be filtered to show Warnings, Errors, or both.

3.4.1 SIMULATION

System-level testing may be performed with ISIM or the ModelSim logic simulator, and such test programs must also be written in HDL languages.[3] Test bench programs may include simulated input signal waveforms, or monitors which observe and verify the outputs of the device under test.[3]
Model Sim or ISIM may be used to perform the following types of simulations:[4]

- Logical verification, to ensure the module produces expected results
- Behavioral verification, to verify logical and timing issues
- Post-place & route simulation, to verify behavior after placement of the module within the reconfigurable logic of the FPGA

3.4.2 SYNTHESIS

Xilinx's patented algorithms for synthesis allow designs to run up to 30% faster than competing programs, and allows greater logic density which reduces project time and costs.
Also, due to the increasing complexity of FPGA fabric, including memory blocks and I/O blocks, more complex synthesis algorithms were developed that separate unrelated modules into slices, reducing post-placement errors.
IP Cores are offered by Xilinx and other third-party vendors, to implement system-level functions such as digital signal processing (DSP), bus interfaces, networking protocols, image processing, embedded processors, and peripherals. Xilinx has been instrumental in shifting designs from ASIC-based implementation to FPGA-based implementation.

3.4.3 EDITIONS

The Subscription Edition is the licensed version of Xilinx ISE, and a free trial version is available for download. The Web Edition is the free version of Xilinx ISE, that can be downloaded and used for no charge. It provides synthesis and programming for a limited number of Xilinx devices. In particular, devices with a large number of I/O pins and large gate matrices are disabled. The low-cost Spartan family of FPGAs is fully supported by this edition, as well as the family of CPLDs, meaning small developers and educational institutions have no overheads from the cost of development software. License registration is required to use the Web Edition of Xilinx ISE, which is free and can be renewed an unlimited number of times

4 DESIGN

4.1 A BRIEF INTRODUCTION TO FPGA

FPGAs are a form of programmable logic, which offer flexibility in design like software, but with performance speeds closer to Application Specific Integrated Circuits (ASICs).

With the ability to be reconfigured an endless amount of times after it has already been manufactured, FPGAs have traditionally been used as a prototyping tool for hardware designers. However, as growing die capacities of FPGAs have increased over the years and so has their use in reconfigurable computing applications.

FPGA ARCHITECTURE

Physically, FPGAs consist of an array of uncommitted elements that can be interconnected in a general way, and is user-programmable. According to Brown et al. every FPGA must embody three fundamental components (or variations thereof) in order to achieve re-configurability {namely logic blocks, interconnection resources, and I/O cells. Digital logic circuits designed by the user are implemented in the FPGA by partitioning the logic into individual logic blocks, which are routed accordingly via interconnection resources. Programmable switches found throughout the interconnection resources dictate how the various logic blocks and I/O cells are routed together. The I/O cells are simply a means of allowing signals to propagate in and out of the FPGA for interaction with external hardware.

Logic blocks, interconnection resources and I/O cells are merely generic terms used to describe any FPGA, since the actual structure and architecture of these components vary from one FPGA vendor to the next. In particular, Xilinx has traditionally manufactured SRAM-based FPGAs; so called because the programmable resources3 for this type of FPGA are controlled by static RAM cells. The fundamental architecture of Xilinx FPGAs is shown in Figure 3.1. It consists of a two dimensional array of programmable logic blocks, referred to as Configurable Logic Blocks (CLBs). The interconnection resources consist of horizontal and vertical routing channels found respectively between rows and columns of logic blocks. Xilinx's proprietary I/O cell architecture is simply referred to as an Input/output Block (IOB). Note that CLB and routing architectures differ for each generation and family of Xilinx FPGA. For example, Figure 3.2 shows the architecture of a CLB from the Xilinx Virtex-E family of FPGAs, which contains four logic cells (LCs) and is organized in two similar slices. Each LC includes a 4-input look-up table (LUT), dedicated fast carry-look ahead logic for arithmetic functions, and a storage element (i.e. a flip-flop). A CLB from the Xilinx Virtex-II family of FPGAs, on the other hand, contains over twice the amount of logic as a Virtex-E CLB. It turns out that the Virtex-II CLB contains four slices, each of which contain two 4-input LUTs, carry logic, arithmetic logic gates.As we will see, the discrepancies in CLB architecture from one family to another is an important factor to take into consideration when comparing the spatial requirements (in terms of CLBs) for circuit designs which have been implemented on different Xilinx FPGAs.

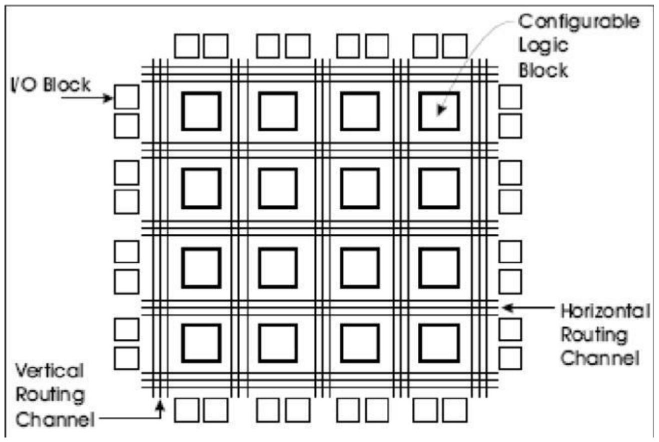


Figure 4.1: General Architecture of Xilinx FPGAs.

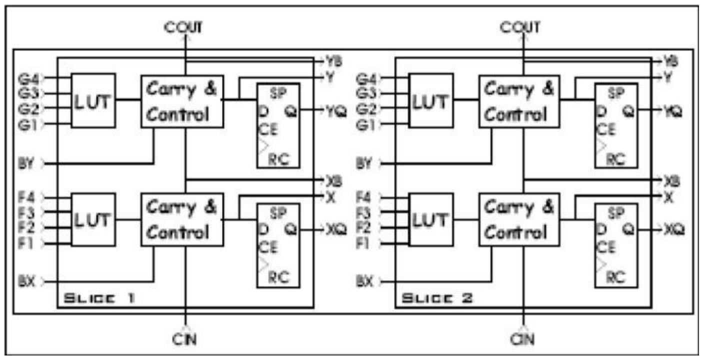


Figure 4.2: Virtex-E Configurable Logic block

4.2 IMPLEMENTATION ISSUES

FPGA implementations of neural networks can be classified on the basis of:

1. Learning Algorithm
2. Signal representation
3. Multiplier reduction schemes

4.2.1 LEARNING ALGORITHM

The type of neural network used in FPGA-based implementations is an important feature used in classifying such architectures. The type of neural network applied depends on the intended application used to solve the problem at hand.

A. Backpropagation Algorithm

1. RRANN Architecture by Elderedge
2. RENCO by Beuchat
3. ACME by Ferucci and Martin
4. ECX card by Skrbeck

B. Ontogenic Neural Networks

FAST Architecture by Perez-Uribe. FAST was used to implement three different kinds of unsupervised, ontogenic neural networks, adaptive resonance theory (ART), adaptive heuristic critic (AHC), and Dyna-SARSA.

C. Cellular Automata based neural networks

CAM- Brain Machine by de Garris.

D. Modular Neural Networks

REMAP Architecture by Nordstrom.

4.2.2 SIGNAL REPRESENTATION

Four common types of signal representations as shown in Figure 3.3, typically seen in ANN h/w architectures are:

Frequency-based - is categorized as a time-dependent signal representation, since it counts the number of analog spikes (or digital 1's depending on h/w medium used) in a given time window (I.e. of n clock cycles). It is popular in analog hardware implementations.

Spike Train - is categorized as a time- and space-dependent signal representation, since the information it contains is based on spacing between spikes (1's) and is delivered in the form of a real number (or integer) within each clock cycle. Used in the CBM.

Floating-point - is considered to be position-dependent because numerical values are represented as strings of digits. Floating-point as a signal representation for FPGA- based (i.e. digital) ANN architectures has been deemed as overkill. This is due to the fact that valuable circuit area is wasted in providing an over-abundance of range-precision, which is never fully utilized by most ANN applications.

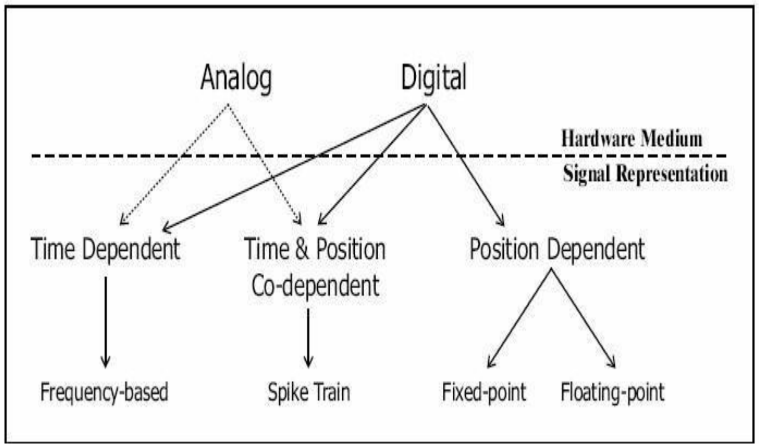


Figure 4.3 Signal Representation

Fixed-point - is categorized as yet another position-dependent signal representation. Fixed-point is the most popular signal representation used among all the surveyed FPGA-based (i.e. digital) ANN architectures. This is due to the fact that fixed-point has traditionally been more area efficient than floating-point, and is not as severely limited in range-precision as both, frequency and spike-train signal representations:

The issues related to fixed point signal representation are

1. Overflow and underflow
2. Convergence rates
3. Quality of generalization

One way to help achieve the density advantage of reconfigurable computing over general purpose computing is to make the most efficient use of the hardware area available. In terms of an optimal range-precision vs area trade-off, this can be achieved by determining the minimum allowable precision and minimum allowable range, where their criterion is to minimize hardware area usage without sacrificing quality of performance. These two concepts combined can also be referred to as the minimum allowable range-precision. Holt and Baker showed that 16-bit fixed point was the minimum allowable range-precision for the back propagation algorithm. However, minimizing range precision (i.e. maximizing processing density) without affecting convergence rates is applications-specific, and must be determined empirically.

4.2.3 MULTIPLIER REDUCTION SCHEMES

The multiplier has been identified as the most area-intensive arithmetic operator used in FPGA based ANNs.

Use of bit-serial multipliers - This kind of digital multiplier only calculates one bit at a time, whereas a fully parallel multiplier calculates all bits simultaneously. Hence, bit-serial can scale up to a signal representation of any range-precision, while its area-efficient hardware implementation remains static. However, the time vs. area trade-off of bit-serial means that multiplication time grows quadratic ally, with the length of signal representation used. Use of pipelining is one way to help compensate for such long multiplication times, and increase data throughput.

Reduce range-precision of multiplier - achieved by reducing range-precision of signal representation used in (fully parallel-bit) multiplier. Unfortunately, this is not a feasible approach since limited range-precision has a negative effect on convergence rates, as discussed in previous section.

Signal representations that eliminate the need for multipliers - Certain types of signal representations replace the need of multipliers with a less area-intensive logic operator. Perez-Uribe considered using a stochastic-based spike train signal his FAST neuron architecture, where multiplication of two independent signals could be carried out using a two-input logic gate. Nordstrom implemented a variant of REMAP for use with Sparse Distributed Memory ANN types, which allowed each multiplier to be replaced by a counter preceded by an XOR logic gate. Another approach would be to limit values to powers of two, thereby reducing multiplications to simple shifts that can be achieved in hardware using barrel shifters.

5 IMPLEMENTATION

5.1 A BRIEF INTRODUCTION TO ANN

Artificial neural networks (ANNs) are a form of artificial intelligence, which have been modeled after, and inspired by the processes of the human brain. Structurally, ANNs consist of massively parallel, highly interconnected processing elements. In theory, each processing element, or neuron, is far too simplistic to learn anything meaningful on its own. Significant learning capacity, and hence, processing power only comes from the culmination of many neurons inside a neural network (as given in Figure 5.1). The learning potential of ANNs has been demonstrated in different areas of application, such as pattern recognition, function approximation/prediction, and robot control

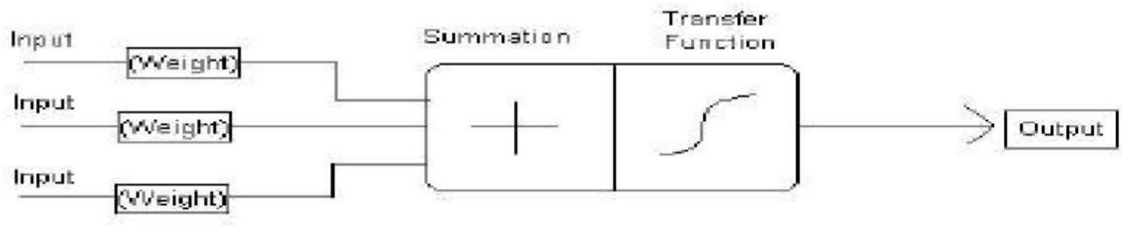


Figure 5.1 The Perceptron Model

Back propagation Algorithm

ANNs can be classified into two general types according to how they learn – supervised or unsupervised. The back-propagation algorithm is considered to be a supervised learning algorithm, which requires a trainer to provide not only the inputs, but also the expected outputs. Unfortunately, this places added responsibility on the trainer to determine the correct input/output patterns of a given problem a priori. Unsupervised ANNs do not require the trainer to supply the expected outputs.

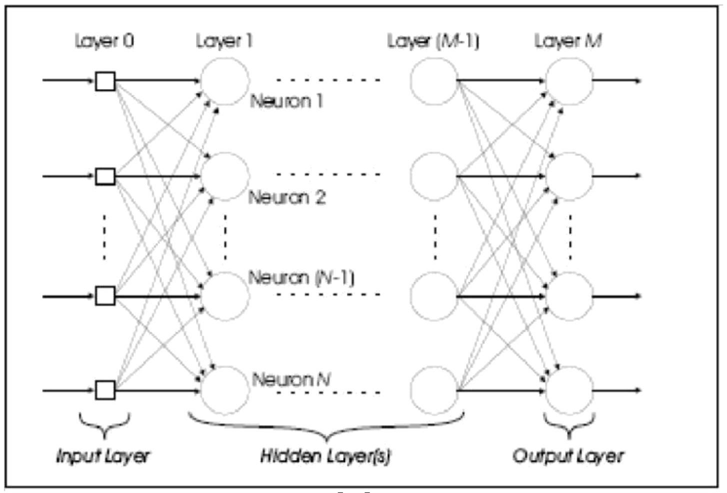


Figure 5.2 Multilayer Perceptron

According to Rumelhart et al. an ANN using the back propagation algorithm has five steps of execution:

Initialization- The following initial parameters have to determine by the ANN trainer a priori:

- $w^{(s)}kj\ (n)$ is defined as the synaptic weight that corresponds to the connection from neuron unit j in the (s - 1)th layer, to k in the sth layer of the neural network. This weight was calculated during the nth iteration of the back propagation, where n = 0 for initialization.
- η is defined as the learning rate and is a constant scaling factor used to control the step size in error correction during each iteration of the back propagation algorithm. Typical values of η range from 0.1 to 0.5.
- $\theta^{(s)}k$ is defined as the bias of a neuron, which is similar to synaptic weight in that it corresponds to a connection to neuron unit k in the sth layer of the ANN, but is NOT connected to any neuron unit j in the (s - 1)th layer. Statistically, biases can be thought of as noise, which better randomizes initial conditions, and increases the chances of
  convergence for an ANN. Typical values of $\theta^{(s)}k$ are the same as those used for synaptic weights $w^{(s)}kj(n)$ in a given application.

Presentation of Training Examples- Using the training data available, present the ANN with one or more *epoch*. An epoch, as defined by Haykin, is one complete presentation of the entire training set during the learning process. For each training example in the set, perform forward followed by backward computations consecutively.

set during the learning process. For each training example in the set, perform forward followed by backward computations consecutively.

Forward Computation- During the forward computation, data from neurons of a lower layer (i.e.(s-1)th layer), are propagated forward to neurons in the upper layer (i.e. sth layer) via a feed-forward connection network. The structure of such a neural network is shown in Figure 4.1, where layers are numbered 0 to M, and neurons are numbered 1 to N. The computation performed by each neuron during forward computation is as follows:

$$H_k^{(s)} = \sum_{j=1}^{N_{s-1}} w_{kj}^{(s)} o_j^{(s-1)} + \theta_k^{(s)}$$

Equation 5.1

where $j < k$ and $s = 1; : : : ;M$

$H^{(s)}k$ = weighted sum of the kth neuron in the sth layer $w^{(s)}kj$ = synaptic weight which corresponds to the connection from neuron unit j in the $(s - 1)^{th}$ layer to neuron unit k in the s^{th} layer of the neural network $o^{(s-1)}j$ = neuron output of the j^{th} neuron in the $(s - 1)^{th}$ layer $\theta^{(s)}k$= bias of the k^{th} neuron in the s^{th} layer

$$o_k^{(s)} = f(H_k^{(s)})$$

Equation 5.2

where $k = 1; : : : ;N$ and $s = 1; : : : ;M$ $o^{(s)}k$ = neuron output of the k^{th} neuron in the s^{th} layer $f(H^{(s)}k\ )$ = activation function computed on the weighted sum $H^{(s)}k$

Note that some sort of sigmoid function is often used as the nonlinear activation function, such as the logsig function shown in the following:

Equation 5.3

$$f(x)_{logsig} = \frac{1}{1+\exp(-x)}$$

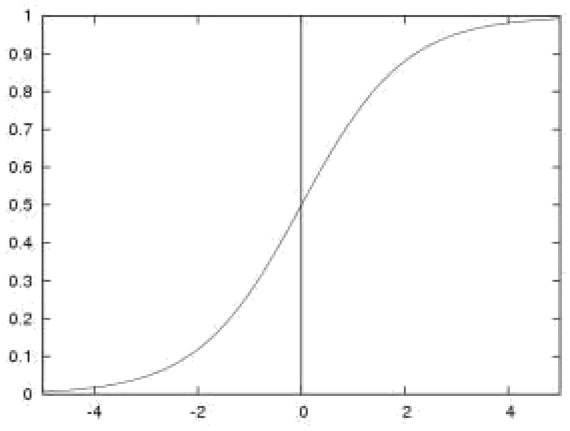


Figure 5.3 The Sigmoid Function

Backward Computation- The back propagation algorithm is executed in the backward computation, although a number of other ANN training algorithms can just as easily be substituted here. Criterion for the learning algorithm is to minimize the error between the expected (or teacher) value and the actual output value that was determined in the Forward Computation. The back propagation algorithm is defined as follows:

1. Starting with the output layer, and moving back towards the input layer, calculate the local gradients, as shown in Equations 5.4., 5.5, and 5.6. For example, once all the local gradients

   are calculated in the s th layer, use those new gradients in calculating the local gradients in the (s -1)th layer of the ANN. The calculation of local gradients helps determine which connections in the entire network were at fault for the error generated in the previous Forward Computation, an is known as error credit assignment.

2. Calculate the weight (and bias) changes for all the weights using Equation 5.7.
3. Update all the weights (and biases) via Equation 5.8.

$$\varepsilon_k^{(s)} = \begin{cases} t_k - o_k^{(s)} & s = M \\ \sum_{j=1}^{N_{s+1}} w_{kj}^{s+1} \delta_j^{(s+1)} & s = 1, \ldots, M-1 \end{cases}$$ Equation 5.4

, where $\varepsilon^{(s)}{}_k$ = error term for the kth neuron in the sth layer; the difference between the teaching signal t_k and the neuron output $o^{(s)}{}_k$ $\delta^{(s+1)}{}_j$ = local gradient for the jth neuron in the (s + 1)th layer.

$$\delta_k^{(s)} = \varepsilon_k^{(s)} f'(H_k^{(s)}) \quad s = 1, \ldots, M$$ Equation 5.5

, where $f'(H^{(s)}k)$ is the derivative of the activation function , which is actually a partial derivative of activation function w.r.t net input (i.e. weight sum), or

$$f'(H_k^{(s)}) = \frac{\partial(a_k^{(s)})}{\partial(H_k^{(s)})} = (1 - a_k^{(s)})a_k^{(s)} \quad \text{for logsig function}$$ Equation 5.6

, where $a^{(s)}k = f(H^{(s)}k) = o^{s}k$

$$\Delta w_{kj}^{(s)} = \eta \delta_k^{(s)} o_j^{(s-1)} \quad k = 1, \ldots, N_s \quad j = 1, \ldots, N_{s-1}$$ Equation 5.7

, where $\Delta w^{(s)}kj$ is the change in synaptic weight (or bias) corresponding to the gradient of error for connection from neuron unit j in the (s - 1)th layer, to neuron k in the sth layer.

$$w_{kj}^{s}(n+1) = \Delta w_{kj}^{(s)}(n) + w_{kj}^{(s)}(n)$$ Equation 5.8

, where k = 1; : : : ;Ns and j = 1; : : : ;Ns-1

$w^{s}kj$ (n + 1) = updated synaptic weight (or bias) to be used in the (n + 1)th
iteration of the Forward Computation

$\Delta w^{(s)}kj$ (n) = change in synaptic weight (or bias) calculated in the nth iteration of the Backward Computation, where n = the current iteration $w^{(s)}kj$ (n) = synaptic weight (or bias) to be used in the nth iteration of the Forward and Backward Computations, where n = the current iteration.

$$\delta_i = y_i(1 - y_i)(d_i - y_i)$$

$$\delta_p(q) = x_p(q)[1 - x_p(q)] \sum w_{p+1}(q,i)\delta_{p+1}(i)$$

Iteration- Reiterate the Forward and Backward Computations for each training example in the epoch. The trainer can continue to train the ANN using one or more epoch until some stopping criteria (eg. low error) is met. Once training is complete, the ANN only needs to carry out the Forward Computation when used in application.

5.2 IMPLEMENTATION APPROACHES

Basically when implementing the back propagation algorithms on FPGAs there are two approaches

1. Non-RTR Approach
2. RTR Approach

RTR stands for Run-Time-Reconfiguration.

5.2.1 NON-RTR APPROACH

The salient features of this approach are:

- All the stages of algorithm reside inside the FPGA at once.
- For the back propagation algorithm the hardware elements required are adders, subtracters, multipliers and transfer function implementers.
- A finite state machine oversees the sequential execution of the stages using the a fore mentioned hardware elements.
- The key design factor is the efficient use of the limited FPGA resources i.e. efficient design of the hardware components.

Performance enhancement can be achieved through:

1. Proper selection of parameters like range-precision.
2. Efficient design of the hardware components used.
3. Pipelining

Floating point representations have highest range-precision but require the largest area on chip. Fixed point is more area efficient and can provide acceptable range-precision.

Holt and Baker showed that 16-bit fixed-point was the minimum allowable range-precision for the back propagation algorithm. However, minimizing range precision (i.e. maximizing processing density) without affecting convergence rates is application-specific, and must be determined empirically.

One challenge in implementing the back propagation on FPGAs is the sequential nature of processing between layers. A major challenge is that pipelining of the algorithm on a whole cannot occur during training due to the weight update dependencies of back propagation, and as a result the utilization of hardware resources is wasted.

However, it's still possible to use fine-grain pipelining in each of the individual arithmetic functions of the back propagation algorithm, which could help increase both, data throughput and global clock speeds

Problems with this approach

- Inefficient use of hardware resources because of the presence of idle circuitry during all times.
- Does not fully justify the use of an FPGA except for prototyping.
- No scalability in topology.
- Large simulation times (which is not practical) using conventional HDLs.

5.2.2 RTR APPROACH

- Only the stage in current execution is configured onto the chip.
- The back propagation algorithm can be divided into three stages of execution –

Feed forward, backward computation and weight update.

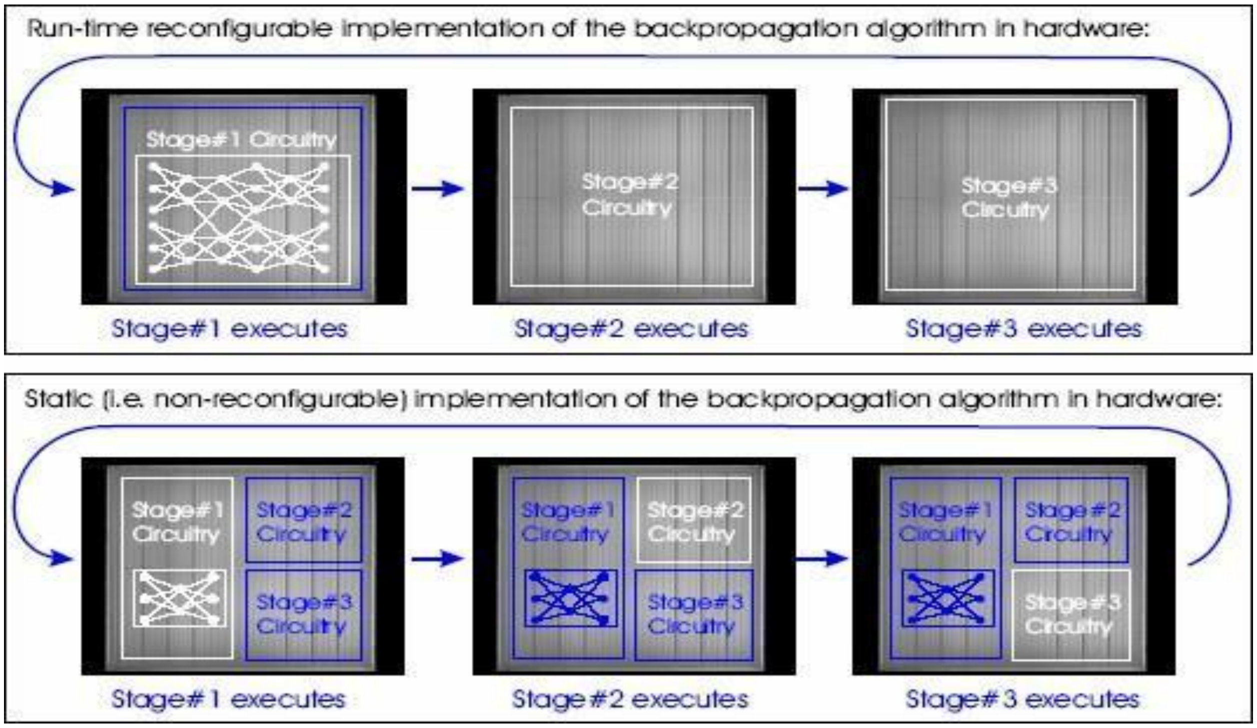


Figure 5.4 RTR and NON- RTR approach

Reconfiguration

FPGA support reconfiguration or programmability but at the cost of reduced performance as compared to ASICs. Therefore to justify the use of FPGAs in ANNs we have to exploit this re-configurability. Thus through reconfiguration the functional density of the FPGA is increased. Run-time-reconfiguration will involve reprogramming time overhead. For networks with a small number of neurons the non RTR implementation is better but for networks with upwards of 23 neurons the RTR method gives better results.

5.3 INTRODUCTION TO XOR PROBLEM

The XOR logic function has two inputs and one output. It produces an output only if either one or the other of the inputs is on, but not if both are off or both ate on. We can consider this as a problem that we want the perceptron to learn to solve: output a 1 if the X is on and **Y** is off, or if **Y** is on and **X** is off, otherwise output 0. It appears to be a simple enough problem. The failure of the perceptron to successfully solve apparently simple problems such as the XOR one was first demonstrated by Minsky and Papert in their influential book Perceptrons.

The XOR problem demonstrates some of the difficulties associated with learning in multilayer perceptrons. Occasionally the network could move in the energy landscape, to cross before reaching an actual deeper minimum, but the network has no way of knowing this, since learning is accomplished by following the energy function down it the steepest direction, until it reaches the bottom of a well, at which point there is no direction to move in order to reduce the energy. There are alternative ways to minimize these occurrences.

If the rate at which the weights are altered is progressively decreased, then the gradient descent algorithm is able to achieve a better solution.
Local minima can be considered to occur when two or more disjoint classes are categorized as the same. This amounts to poor internal representation within the hidden units, and so adding more units to this layer will allow a better recoding of the inputs and lessen the occurrence of these minima.

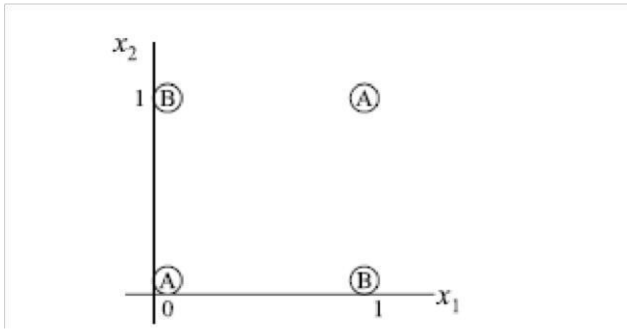


Figure 5.5 The exclusive OR (XOR) problem: points (0,0) and (1,1) are members of class A; points (0,1) and (1,0) are members of class B.

5.4 IMPLEMENTATION APPROACH

The XOR Problem was implemented using a 3 layer, 5 neuron, multilayer perceptron model of Artificial Neural Network. The following approach was followed: -

1. The network was initialized and random values were given to the weights (between -1 and 1)
2. The weighted sum of the input values at the hidden neurons was calculated.
3. The obtained value was run through a hard limiter function.
4. The same procedure was followed for the output layer using the outputs of the hidden layer as the input.
5. The value for deltas for the output and hidden layer was calculated.
6. The weights were then updated using the delta values.
7. The learning parameter, η, was taken as 0.5

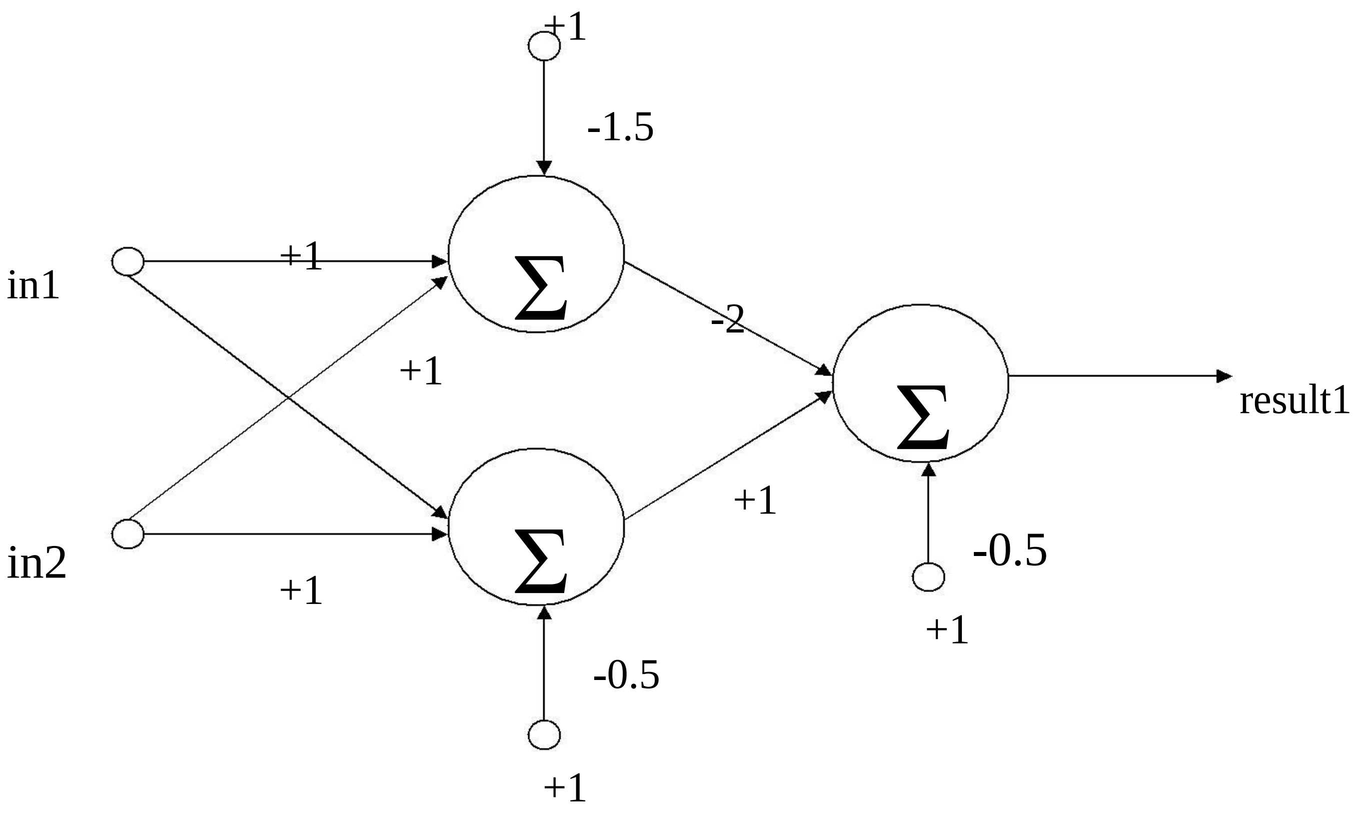
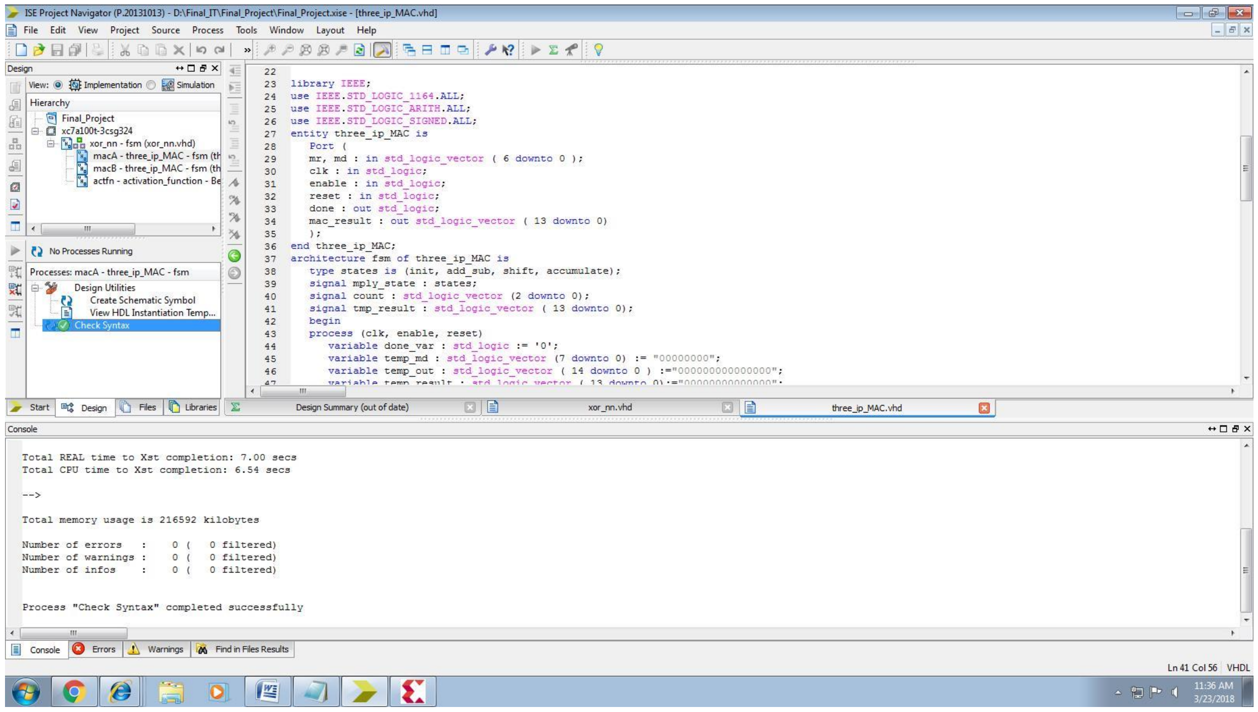


Figure 5.6 Neuron model for XOR Problem

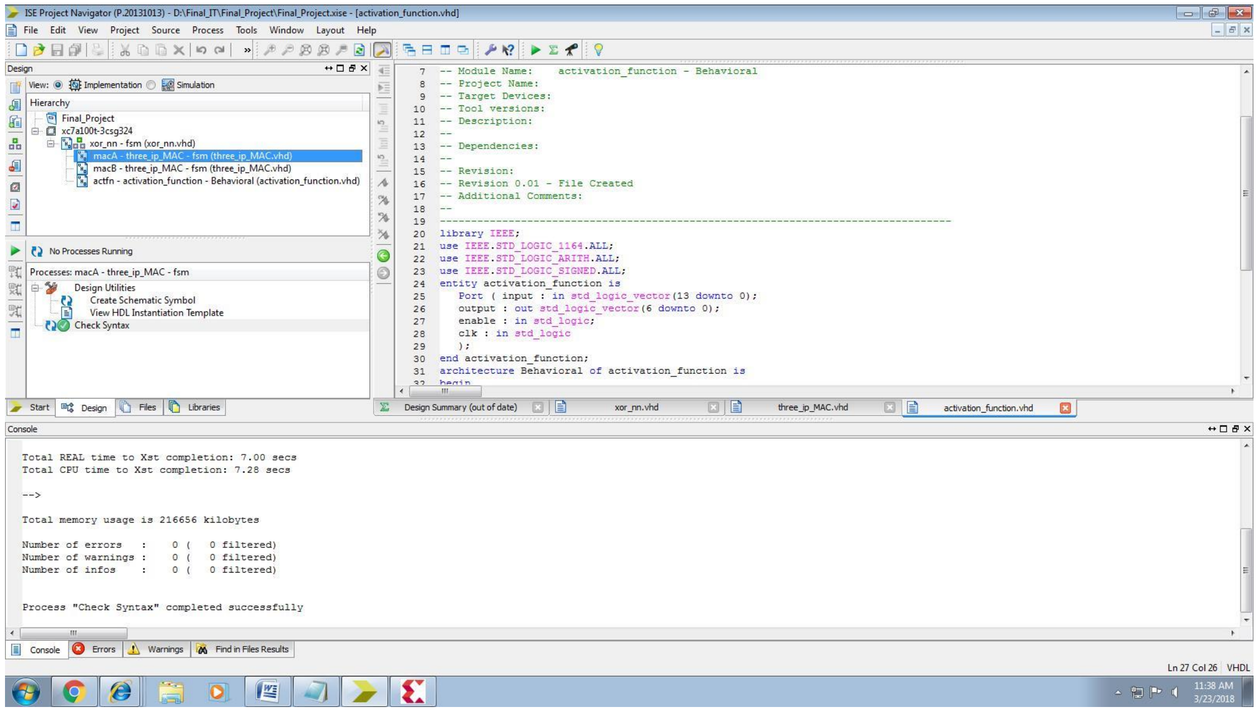
6. TESTING AND RESULT


ISE Project Navigator (P.20131013) - D:\Final_IT\Final_Project\Final_Project.xise - [three_ip_MAC.vhd]
Hierarchy
Final_Project
xc7a100t-3csg324
xor_nn - fsm (xor_nn.vhd)
macA - three_ip_MAC - fsm
macB - three_ip_MAC - fsm
actfn - activation_function
No Processes Running
Processes: macA - three_ip_MAC - fsm
Design Utilities
Create Schematic Symbol
View HDL Instantiation Temp...
Check Syntax
Design Summary (out of date)
xor_nn.vhd
three_ip_MAC.vhd
Ln 41 Col 56 VHDL
11:36 AM
3/23/2018

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity three_ip_MAC is
   Port (
   mr, md : in std_logic_vector ( 6 downto 0 );
   clk : in std_logic;
   enable : in std_logic;
   reset : in std_logic;
   done : out std_logic;
   mac_result : out std_logic_vector ( 13 downto 0)
   );
end three_ip_MAC;
architecture fsm of three_ip_MAC is
   type states is (init, add_sub, shift, accumulate);
   signal mply_state : states;
   signal count : std_logic_vector (2 downto 0);
   signal tmp_result : std_logic_vector ( 13 downto 0);
   begin
   process (clk, enable, reset)
      variable done_var : std_logic := '0';
      variable temp_md : std_logic_vector (7 downto 0) := "00000000";
      variable temp_out : std_logic_vector ( 14 downto 0 ) :="000000000000000";
      variable temp_result : std_logic_vector ( 13 downto 0):="00000000000000";
```

```
Total REAL time to Xst completion: 7.00 secs
Total CPU time to Xst completion: 6.54 secs

-->

Total memory usage is 216592 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    :    0 (   0 filtered)


Process "Check Syntax" completed successfully
```

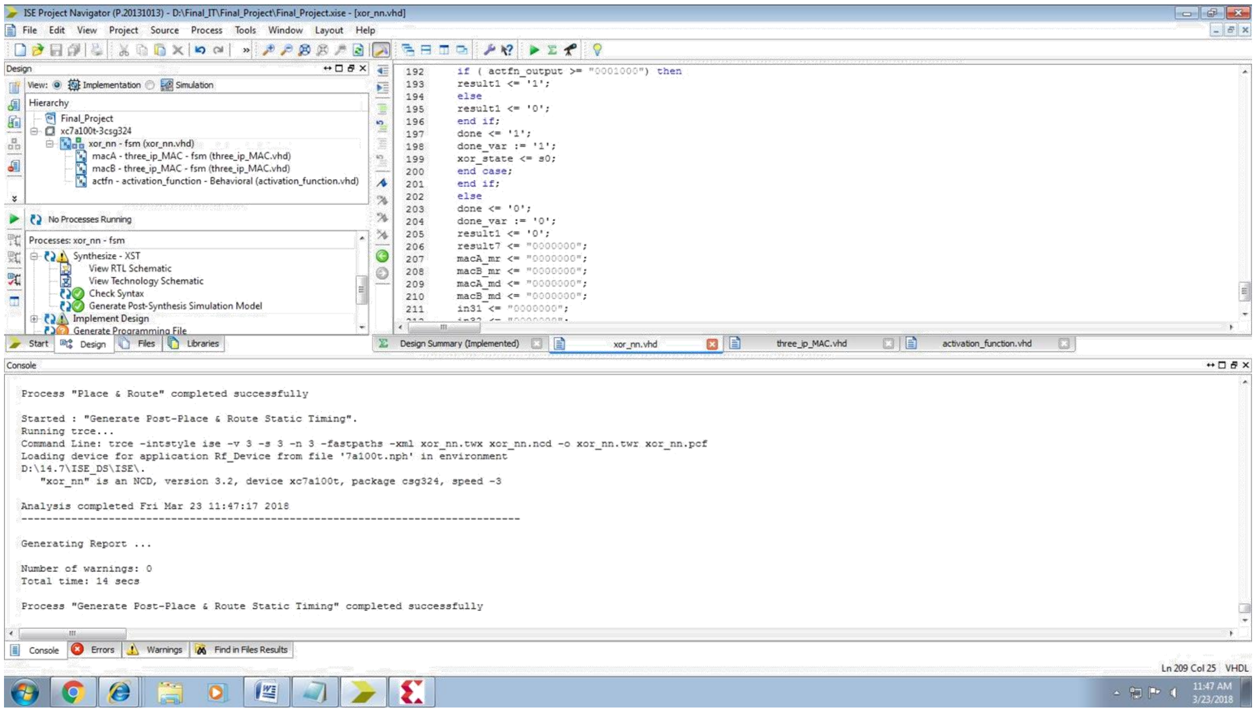
Figure 6.1 : IP MAC block execution


ISE Project Navigator (P.20131013) - D:\Final_IT\Final_Project\Final_Project.xise - [activation_function.vhd]
Hierarchy
Final_Project
xc7a100t-3csg324
xor_nn - fsm (xor_nn.vhd)
macA - three_ip_MAC - fsm (three_ip_MAC.vhd)
macB - three_ip_MAC - fsm (three_ip_MAC.vhd)
actfn - activation_function - Behavioral (activation_function.vhd)
No Processes Running
Processes: macA - three_ip_MAC - fsm
Design Utilities
Create Schematic Symbol
View HDL Instantiation Template
Check Syntax
Design Summary (out of date)
xor_nn.vhd
three_ip_MAC.vhd
activation_function.vhd
Ln 27 Col 26 VHDL
11:38 AM
3/23/2018

```vhdl
-- Module Name:    activation_function - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity activation_function is
   Port ( input : in std_logic_vector(13 downto 0);
   output : out std_logic_vector(6 downto 0);
   enable : in std_logic;
   clk : in std_logic
   );
end activation_function;
architecture Behavioral of activation_function is
begin
```

```
Total REAL time to Xst completion: 7.00 secs
Total CPU time to Xst completion: 7.28 secs

-->

Total memory usage is 216656 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    :    0 (   0 filtered)


Process "Check Syntax" completed successfully
```

Figure 6.2: XOR state function block execution



```
if ( actfn_output >= "0001000") then
result1 <= '1';
else
result1 <= '0';
end if;
done <= '1';
done_var := '1';
xor_state <= s0;
end case;
end if;
else
done <= '0';
done_var := '0';
result1 <= '0';
result7 <= "0000000";
macA_mr <= "0000000";
macB_mr <= "0000000";
macA_md <= "0000000";
macB_md <= "0000000";
in31 <= "0000000";
```

```
Process "Place & Route" completed successfully

Started : "Generate Post-Place & Route Static Timing".
Running trce...
Command Line: trce -intstyle ise -v 3 -s 3 -n 3 -fastpaths -xml xor_nn.twx xor_nn.ncd -o xor_nn.twr xor_nn.pcf
Loading device for application Rf_Device from file '7a100t.nph' in environment
D:\14.7\ISE_DS\ISE\.
   "xor_nn" is an NCD, version 3.2, device xc7a100t, package csg324, speed -3

Analysis completed Fri Mar 23 11:47:17 2018
--------------------------------------------------------------------------------

Generating Report ...

Number of warnings: 0
Total time: 14 secs

Process "Generate Post-Place & Route Static Timing" completed successfully
```

ISE Project Navigator (P.20131013) - D:\Final_IT\Final_Project\Final_Project.xise - [xor_nn.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
Final_Project
xc7a100t-3csg324
xor_nn - fsm (xor_nn.vhd)
macA - three_ip_MAC - fsm (three_ip_MAC.vhd)
macB - three_ip_MAC - fsm (three_ip_MAC.vhd)
actfn - activation_function - Behavioral (activation_function.vhd)
No Processes Running
Processes: xor_nn - fsm
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation Model
Implement Design
Generate Programming File
Start
Design
Files
Libraries
Design Summary (Implemented)
xor_nn.vhd
three_ip_MAC.vhd
activation_function.vhd
Console
Errors
Warnings
Find in Files Results
Ln 209 Col 25 VHDL
11:47 AM
3/23/2018

Figure 6.3: Activation function execution

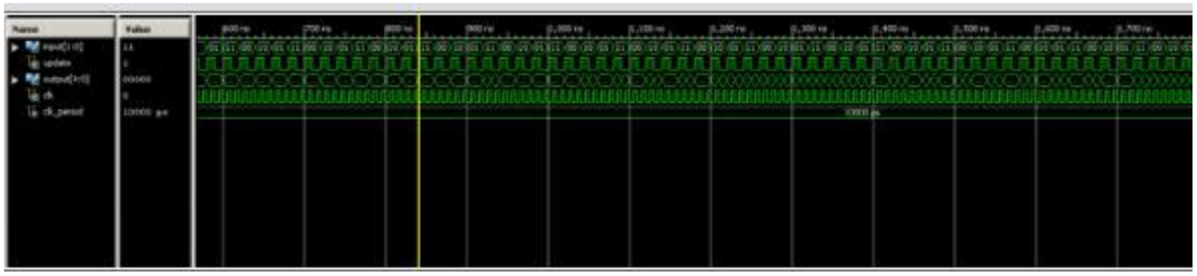


Figure 6.4: Initial Stage output

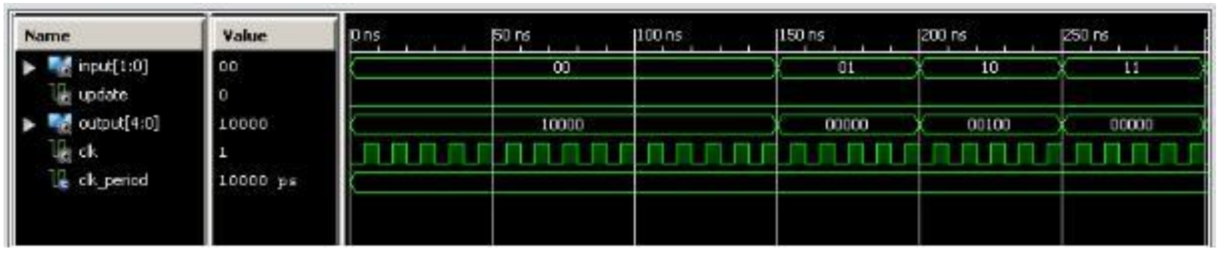
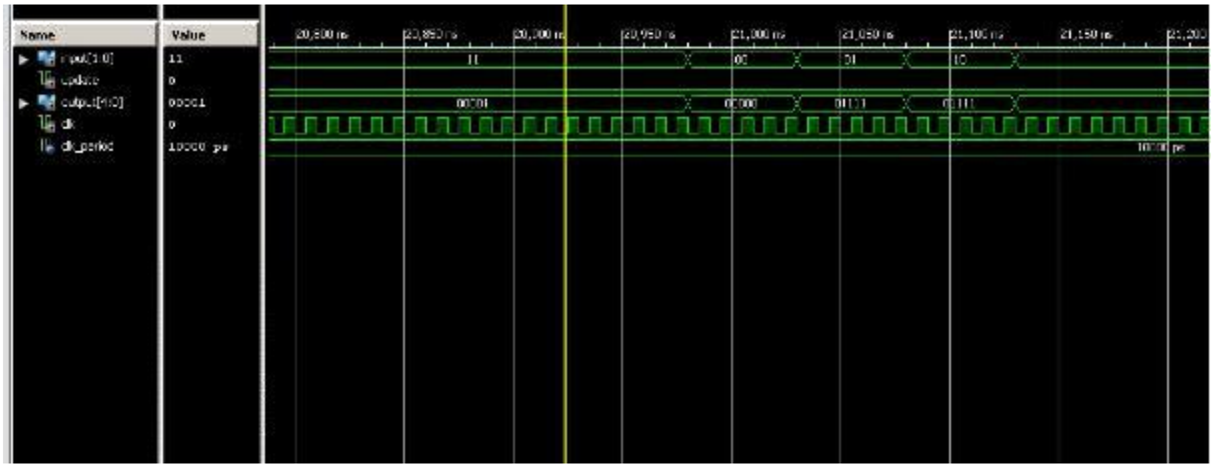


Figure 6.5: Training output



Name	Value
input[1:0]	11
update	0
output[4:0]	00001
clk	0
clk_period	10000 ps

Figure 6.6: Final stage output

HARDWARE UTILIZATION SUMMARY

Device Utilization Summary:

Slice Logic Utilization:

Number of Slice Registers:	137 out of 126,800 1%
Number used as Flip Flops:	137
Number used as Latches:	0
Number used as Latch-thrus:	0
Number used as AND/OR logics:	0
Number of Slice LUTs:	242 out of 63,400 1%
Number used as logic:	232 out of 63,400 1%
Number using O6 output only:	208
Number using O5 output only:	0
Number using O5 and O6:	24
Number used as ROM:	0
Number used as Memory:	0 out of 19,000 0%
Number used exclusively as route-thrus:	10
Number with same-slice register load:	4

```
    Number with same-slice carry load:        6
    Number with other load:                 0

Slice Logic Distribution:
    Number of occupied Slices:              106 out of 15,850  1%
    Number of LUT Flip Flop pairs used:     260
    Number with an unused Flip Flop:         133 out of     260 51%
    Number with an unused LUT:                18 out of     260    6%
    Number of fully used LUT-FF pairs:       109 out of     260     41%
    Number of slice register sites
    lost to control set restrictions:          0 out of 126,800   0%
```

A LUT Flip Flop pair for this architecture represents one LUT paired with one Flip Flop within a slice. A control set is a unique combination of clock, reset, set, and enable signals for registered element.

The Slice Logic Distribution report is not meaningful if the design is over-mapped for a non-slice resource or if Placement fails.

OVERMAPPING of BRAM resources should be ignored if the design is over-mapped for a non-BRAM resource or if placement fails.

7 CONCLUSION AND FUTURE SCOPE

The basic behavior of the biological neuron can be emulated in an artificial neuron. A biological neuron with their dendrites, soma and axon can be characterized in an artificial neuron as a black box with inputs and an output. To implement the system the electronic pulses or spikes transmitted through neurons are replaced by digital signals or pulses. We can get to emulate the potential actions in the pre-synaptic and postsynaptic reactions using timers and multiplier blocks. With all these things we get an electronic system that reproduces the behavior of the biological neuron. The aim is to have the possibility of interconnect more of these artificial neurons to create a complete neuronal network. The motivation for this study stems from the fact that an FPGA coprocessor with limited logic density and capabilities can used in building Artificial Neural Network which is widely used in solving different problems. With the VHDL code generated a FPGA can be programmed.

Depending of the capacity of the FPGA used, a larger o less number of neuron can be programmed, depending on the same way of the number of interconnections of each neuron. Therefore, according to the results obtained, we can say that we have designed a system whose performance leads to the achievement of the objectives of the project and at the
Same time could work as the main base to develop future applications.

BIBILIOGRAPHY

[1] Haykins, Simon , Neural Networks – A comprehensive foundation, Delhi, Pearson Prentice Hall India

[2] Bhaskar, J. A VHDL Primer, Delhi, Pearson Prentice Hall India.

[3] Stallings, William, Computer Architecture And Organisation, Delhi, Pearson Prentice Hall India.

[4] Zhu Jihan, Sutton Peter, FPGA Implementations of Neural Networks - a

[5] Survey of a Decade of Progress, School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Queensland 4072, Australia.

[6] Hammerstrom Dan, Digital VLSI for Neural Networks, Department of Electrical and Computer Engineering, OGI School of Science and Engineering, Oregon Health and Science University

[7] Eldredge James G, Hutchings Brad L, Density Enhancement of a Neural Network Using FPGAs and Run Time Reconfiguration, Presented at IEEE Workshop on FPGAs for Custom Computing Machines Napa, CA, April 10-13, 1994, pg 180-188.

[8] Eldredge James G, Hutchings Brad L, RRANN: - The Run Time Reconfiguration Artificial Neural Network, Presented at IEEE Custom Integrated Circuits Conference, San Diego, CA, May 1-4,1994, pg 77-80.

[9] Gadea Rafael, CerdáJoaquín, BallesterFranciso, Mocholí Antonio, Artificial Neural Network Implementation on a single FPGA of a Pipelined On-Line Back propagation, ISSS 2000, Madrid, Spain ,2000 IEEE 1080-1082/00.

[10] Nichols Kristian Robert, A Reconfigurable Architecture for implementing Artificial Neural Networks on a FPGA, The Faculty of Graduate Studies, The

[11] University of Guelph

[12] Arroyo Ledn Marc A., Castro Arnold Ruiz, AscencioRakl R. Leal, An Artificial Neural Network on a Field Programmable Gate Array as a virtual sensor, 0-7803-5588-1/99, 1999 IEEE Transactions on Neural Networks.

[13] Hadley J. D., Hutchings B. L., Design Methodologies for Partially Reconfigured Systems, 0-8186-7086-X/95, 1995 IEEE, *This work was supported by ARPA/CSTO under contract number DABT63-94-C-0085 under a subcontract to National Semiconductor.

[14] MedhatXiaoguang LI, Areibi Moussa Shawki, Arithmetic formats for implementing Artificial Neural Networks on FPGAs, School of Engineering, University of Guelph, Guelph, ON, CANADA, N1G 2W1

[15] BotrosNaleih M., Arir M. Abdul, Hardware Implementation of an Artificial Neural Network Using Field Programmable Gate Arrays (FPGA's), 0278-0046/94, 1994 IEEE.

[16] Merchant Saumil, Peterson Gregory D., Park Sang Ki, Kong Seong G., FPGA Implementation of Evolvable Block-based Neural Networks, 2006 IEEE Congress on Evolutionary Computation, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada, July 16-21, 2006, 0-7803-9487-9/06, 2006 IEEE

[17] BemleyJessye, Neural Networks and the XOR Problem, 0-7803-7044-9/01/$10.00 02001 IEEE.

[18] Rogers Alan, Keating John G, Shorten Robert, Heffernan Daniel M., Chaotic maps and pattern recognition – the XOR problem, 0960-0779/02, 2002 Elsevier Science Ltd.

[19] Lau TszHei, Implementation of Artificial Neural Network on FPGA Devices, Department of Computer System Engineering, University of Auckland, New Zealand.

[20] Chan Ian D, Implementation of Artificial Neural Network on a FPGA Device, Department of Electrical and Computer Engineering, University of Auckland, Auckland, New Zealand.

[21] Hernández Miriam Galindo, AscencioRaúl R. Leal, Galicia Cuauhtemoc Aguilera, The study of a prototype of a Artificial Neural Network on a Field Programmable Gate Array as a function approximator, ITESO, Departamento de Electrónica, Sistemas e Informática, Tlaquepaque, Jalisc, Mexico.

[22] Alderighi M., Gummati E.L., Phi V, Sechi G.R, A FPGA-based Implementation of a Fault-Tolerant Neural Architecture for Photon Identification, FPGA97, Monterey California USA, 1997 ACM O-89791-801-0/97/02.