Hardware Implementation of Feedforward Multilayer Neural Network Using the RFNNA Design Methodology

Fawnizu Azmadi Hussin[1], Noohul Basheer Zain Ali and Ivan Teh Fu Sun
Universiti Teknologi PETRONAS,
Bandar Seri Iskandar, 31750 Tronoh, Perak Darul Ridzuan, Malaysia.
[1]fawnizu@petronas.com.my

ABSTRACT

This paper proposes a novel hardware architecture for neural network that shall be named Reconfigurable Feedforward Neural Network Architecture (RFNNA) processor [1]. This neural network architecture aims to minimize the logic circuit as required by a fully parallel implementation. The Field-Programmable Gate Array (FPGA)-based RFNNA processor architecture proposed in this paper shared logic circuits for its hidden layer neurons and could be reconfigured for specific applications [2,3], which required different neural network structures. This was achieved by storing connection and neuron weights for the multiple hidden layers in the EPROMs and utilized the hidden layer neuron's logic circuits iteratively for multiplication, summation and evaluation purposes. In this paper, training of neural network was not considered and was performed offline using software. The resulting weights and biases were then loaded into the RFNNA processor's EPROMs for implementation [1]. The RFNNA processor was tested with the XOR non-linear problem using a 2-3-3-3-1 architecture.

Keywords: *Artificial Intelligence, Neural Network Processor, Field Programmable Gate Array, FPGA, Reconfigurable, Hardware Implementation, RFNNA, Feedforward Multilayer*

INTRODUCTION

Neural information processing is an emerging new field, providing an alternative form of computation for demanding tasks. Examples of neural network applications are character recognition and stock market forecasting [4]. Neural networks can be implemented using either software which is based on a central processing unit or on a dedicated hardware, which can process inputs in parallel and decentralized fashion [5]. The latter is a neural network processor, which can yield tremendously fast computations as required by neural network applications. Due to the heavy computation demands of a neural network, hardware based solutions are preferred.

A neural network is made up of a collection of neurons, which are arranged in layers known as hidden layers. Each neuron is an individual processing unit that performs summation and has an activation function that normalizes the result of the summation before passing the value to other neurons in the subsequent hidden layer. The number of neurons in a layer is arbitrary and the number can be between 2 and 200 neurons or even more depending on the application [6]. The number of hidden layers also depends on the application for which the neural network is designed. The allocation of resources between number of neurons and hidden layers are inversely proportionate.

This paper was presented in the International Conference on Neuro-Computing and Evolving Intelligence, Auckland University of Technology Park, Auckland, New Zealand, 13-15 December 2004.

Figure 1 illustrates a typical feedforward neural network structure with two inputs, two hidden layers with three neurons in each, and one output. The dark circle refers to a deposit of biased values, which is unique for each neuron that it is connected to.

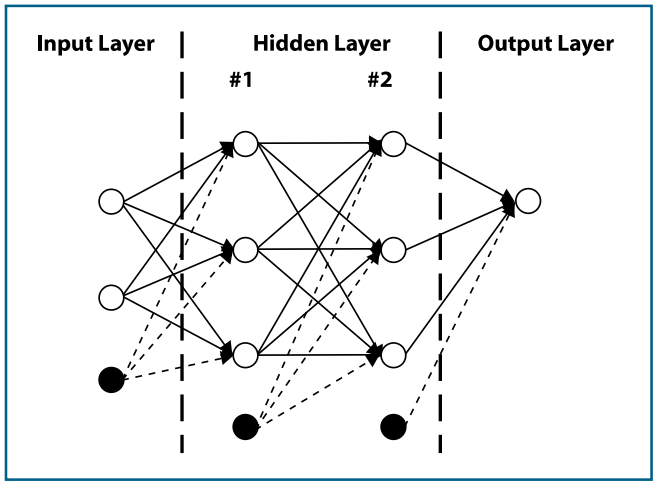

Figure 1: A 2-3-3-1 Neural Network Architecture

Current implementation of a neural network processor requires large amounts of logic resources since it is implemented with a fixed structure, allocating hardware resource for each and every neuron and connection weight.

In order to improve the hardware implementation of the neural network structure in Figure 1, this paper proposes a novel architecture called Reconfigurable Feedforward Neural Network Architecture (RFNNA) based on a paper by Zain Ali, et al. [1]. The RFNNA is a design methodology for hardware implementation of neural networks. The main advantage of using the RFNNA design methodology is that it can reduce the amount of logic gates used for a fully parallel neural network [7,8] of multiple layers to just only one single hidden layer by iterating the use of one hidden layer's resources. Theoretically, a neural network design using RFNNA can have infinite number of hidden layers; though this is constrained by the amount of resources allocated to store connection weights and biases.

ARCHITECTURE OF RFNNA PROCESSOR

Figure 2 below shows a high level view of the RFNNA processor. The Multiplication/Summation Block refered to an individual neuron. There were 3 neurons altogether, catering for the 2-3-3-3-1 architecture, that is, the processor has 2 inputs, 1 output and 3 hidden layers with 3 neurons each. The block diagram as in Figure 2 was broken down into 11 functional modules as in Figure 3 for the actual design. Selected modules are described as follows.

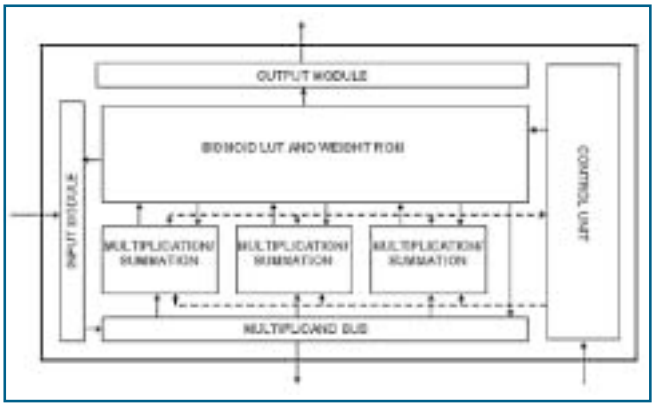

Figure 2: RFNNA Processor for XOR Problem

Input Module

This module served as buffer and multiplexer to external inputs as well as outputs from the activation function. External inputs were first converted into its equivalent value for arithmetic computation. The input module accepts and assigns values to external inputs to be stored into the multiplier buffers concurrently. However the same buffers were written sequentially when data was passed from the activation function. No conversion was required for data from the activation function. Besides receiving and storing data, the Input Module was also required to correctly broadcast the values in its buffers onto the multiplicand bus.

Bias and Weight EPROM Module

There were three Bias and Weight EPROM Modules in the designed processor. Each module was similar to the other, differing only in the weight and bias values

they carried. Each module was dedicated to one neuron and stores bias and weight values for three hidden layers and one output layer. As in the Input Module, weight values were sequentially passed to the Neuron Module. Each module had an internal counter which told the module which weight value is to be passed on. External signals to the modules told which layer the bias and weights value it could select from. The bias values stored in the modules were in twos complement format while weight values were stored in signed magnitude integer. The fraction size for the weight and bias values were different. The fraction size for weight values was equivalent to 0.01 (11 bit word) whereas the fraction size for bias values was 0.0001 (21 bit word).

Neuron Module

Multiplication for the neuron module used the Add Shift Right (ASR) algorithm. This algorithm was suited for unsigned binary multiplication, which was the type of data presented to it. While the multiplication algorithm involved unsigned binary integer, a register was used to store the sign bit of the multiplicand. The sign bit acts as a flag as to whether the multiplication result needed to be complemented before it is summed up with the bias value stored in a 21 bit wide output register.

There was only one multiplier designed into each neuron, therefore multiplier and multiplicand values provided by the Input Module and the Bias and Weight EPROM Module were multiplied and summed up sequentially with the bias value which was directly stored in the 21 bit wide register. The use of only one multiplier per neuron was justified with the amount of logic saved as compared to the time used to complete iteration per hidden layer. The time factor for multiplication was now prolonged corresponding to the number of neurons N per hidden layer. Comparatively, if a fully parallel implementation were to be used, the number of logic gates for the multiplication portion would be N^2 as much. The loading of values would different for the starting of a hidden layer and the loading of values thereafter. Bias values would only be loaded once for every hidden layer into the Neuron Module whereas multiplier and multiplicand values were loaded during each iteration.

Activation Function LUT Module

The LUT, which was implemented and declared as an EPROM block in the FPGA device itself, contained many redundant entries. Addressing the LUT was an 8-bit input, which had 256 entries. Using mathematical analysis, it was possible to reduce the EPROM usage from 256 x 8 bit word entries to 63 x 8 bit word entries. This was because there were only 50 unique data that was being accessed in the LUT ranging from decimal equivalent of 50 to 99. Several of the combinations could be grouped together for an entry thus reducing the need for individual access for equivalent results.

The inputs from the Number Representation Converter Module were used to address the activation function LUT as well as to note the sign of the input argument. As was mentioned before, the LUT table could only address values from 50 to 99 corresponding to the sigmoid range of 0.50 to 0.99. These values were only valid for positive arguments. If the sign of the argument were negative, some manipulation of the LUT result had to be performed. The LUT equivalent would be subtracted from 100 to produce the correct answer.

Control Unit

The Control Unit for this neural network processor had 10 distinct inputs and 10 distinct outputs. The Control unit was able to guide the rest of the modules to function as intended at the right time.

The Control Unit provided control signals to all modules except for the Number Representation Converter Module and the Activation Function LUT Module. The system was designed such that in case of a reset, the whole processor could be set back to its initial state.

The Ready flag, which was connected to an external output pin, notifies the user whether the processor was available for processing or when processing was being performed.

ORGANIZATION OF RFNNA PROCESSOR

The modules described earlier were arranged and connected as shown in Figure 3. There were a total of 5 external inputs and 2 external outputs.

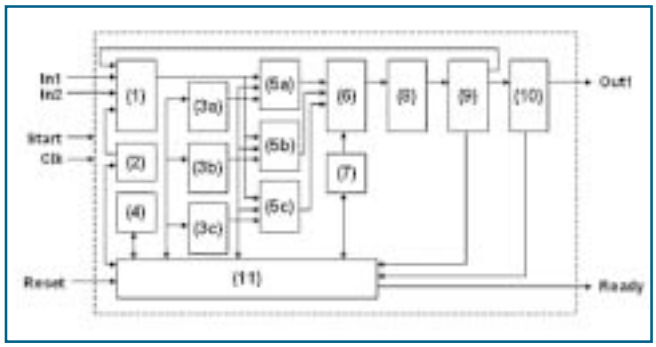

Figure 3: Block diagram for the RFNNA Processor

(1) Input Module
(2) Input Module Counter
(3) Bias and Weight EPROM Module
(4) Bias and Weight Counter Module
(5) Neuron Module
(6) Neuron Output Multiplexer Module
(7) Neuron Output Multiplexer Counter Module
(8) Number Representation Converter Module
(9) Activation Function LUT Module
(10) Output Threshold Module
(11) Control Unit

The Input Module (1) took in inputs from external inputs and also from the output of the Activation Function LUT Module (9). The output from the Input Module was simultaneously fed into the 3 Neuron Modules (5a, 5b, 5c) via a 7-bit wide data bus. Individual Bias and Weight EPROM Modules (3a, 3b, 3c) were assigned to each Neuron Module, providing two types of inputs to it; bias values (21 bit) and weight values (11 bit).

Multiplication of inputs and weights were performed sequentially and the result summed up after each iteration. By employing the ASR multiplication algorithm, all multiplication tasks in the neurons were completed simultaneously, given the same multiplier value (input) regardless of what the multiplicand value (weight) was. Therefore only one of the modules needed to provide feedback to the Control Unit (11) as all other neuron computations were synchronized.

There was only one Activation Function LUT Module. Therefore each Neuron Module output had to take turns to access the activation function resources. This task was performed via the Neuron Output Multiplexer Module (6) which multiplexed between the 3 neurons. The output of the Activation Function LUT Module was directed to 2 modules, Input Module and Output Threshold Module (10).

A neural network's neuron encompasses the summing up of weighted inputs and passing the value through the activation function. However for this design, the individual neuron's function included the multiplication of input values with the connection weights. The passing of summation results for each neuron into the activation function were detached into a separate process.

The Control Unit through the external output Ready notified the user whether the value of the Out pin was valid or otherwise.

EXPANDABILITY OF RFNNA PROCESSOR

Input values were passed to the Neuron Modules through a multiplicand bus (refer Figure 2). This bus when connected to external outputs could be used to expand the capacity of the RFNNA processor. This expandability adds to the versatility of the RFNNA processor making it possible to be used for applications which require more neurons per hidden layer that can be fitted into a single FPGA chip [1]. Figure 4 illustrates how multiple RFNNA processors can be connected together for parallel processing.

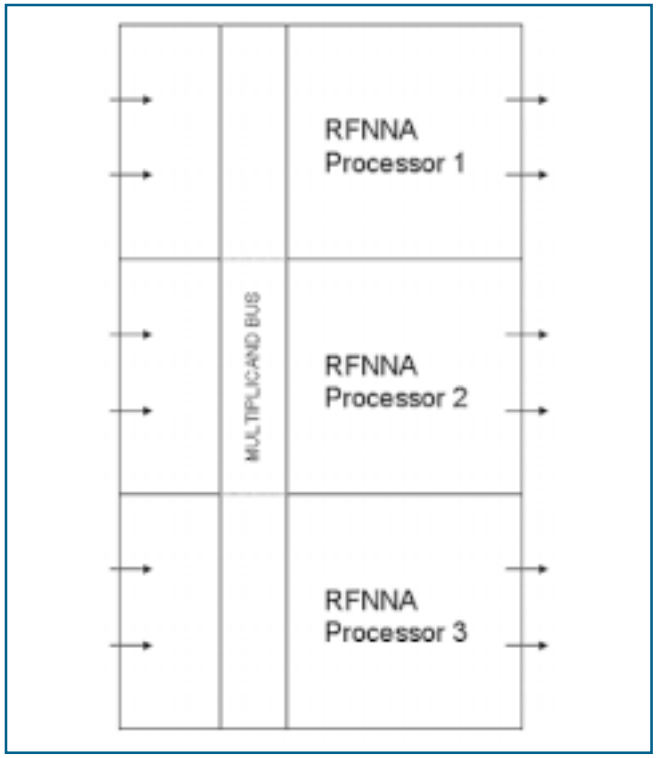

Figure 4: Parallel Processing of RFNNA Processors

CONCLUSION

The paper describes the advantages of a novel hardware architecture for neural network called RFNNA. RFNNA is a design methodology to reduce circuit area requirement in an integrated circuit. The optimization is achieved through reiteration of a single hidden layer. This approach enables the RFNNA processor to cater for applications, which require any number of hidden layer neurons. The versatility of the RFNNA processor is improved by allowing it to be connected in parallel thereby permitting a larger number of neurons per hidden layer.

The weight and bias values are stored within the processor's EPROM itself, thus limiting the number of hidden layer weights and biases that can be stored. For the RFNNA processor to cater for any application in a single chip, the design must have an external RAM to store information for weight, bias and network connectivity information. Thus when the processor is need to be run different applications, only RAM values need to be reinitialized. This general-purpose design would also be suitable for hardware implementation using Application Specific Integrated Circuits (ASICs) which is capable of faster processing speeds and a larger neuron count.

NOMENCLATURE

H	=	Inertia constant
WR^2	=	Wight of rotating parts of generating unit (lb) multiplied by the square of radius of gyration (ft)
RPM	=	Rotation per minute
H_n	=	Inertia constant of n^{th} generating unit
RPM_n	=	Rotation per minute of n^{th} generating unit
kVA	=	Apparent power
P	=	Decelerating power in per-unit of connected kVA
df/dt	=	Rate of frequency decline
GTG	=	Gas turbine generator
STG	=	Steam turbine generator
SS	=	Substation
MISS	=	Main intake substation
CB	=	Circuit breaker
t_{trip}	=	Trip time
$t_{pick-up}$	=	Pick up time

REFERENCES

[1] Zain Ali, N.B., Ivan Teh F.S. 2004, *Reconfiguring Logic Resources of Single Hidden Layer to Accommodate Multiple Layer Neural Network Applications*, in Proceedings of the 2nd International Conference on Artificial Intelligence in Engineering and Technology.

[2] J. Zhu, G.J. Milne and B.K. Gunther, *Towards An FPGA Based Reconfigurable Computing Environment for Neural Network Implementations*, in Proceedings of IEE Conference on Artificial Neural Networks, 1999: p. 661-666.

[3] Eldredge, J.G. and B.L Hutchings, *Design Methodologies for Partially Reconfigured Systems*, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1994: p. 78-84.

[4] Callan R. 1999, Essence of Neural Networks, Hertfosrdshire, Prentice Hall Europe.

[5] Szabo T., Feher B. and Horvath G. 1998, *Neural Network Implementation Using Distributed Arithmetic*, in Proceedings of the 2nd International Conference on Knowledge Based Intelligent Electronic Systems.

[6] Haykin S. 1999, Neural *Networks – A Comprehensive Foundation*, Prentice Hall New Jersey.

[7] Sundararajan N., Saratchandran P. 1998, *Parallel Architectures for Artificial Neural Networks*, IEEE Computer Society, Los Alamitos, California.

[8] Gschwind, M. V. Salapura. and O. Maischberger, *Space Efficient Neural Network Implementation*, in Proceedings of the 2nd ACM Workshop on Field Programmable Gate Arrays, 1994. p. 23-28.

Fawnizu Azmadi Hussin is a faculty member at the Universiti Teknologi PETRONAS, Malaysia. He obtained his BSc in Electrical Engineering, specializing in Computer Design from the University of Minnesota, USA and subsequently his MEngSc in Systems and Control from the University of New South Wales, Australia. His research interests are in VLSI design and testing, FPGA prototyping of neural network architecture, encryption algorithms and simulation of a dynamic system. He is currently pursuing his PhD in VLSI Design for Testability at Nara Institute of Science and Technology, Japan.

Noohul Basheer Zain Ali is a faculty member at the Universiti Teknologi PETRONAS, Malaysia. He obtained his MSc in Computer and Systems Engineering from Rensselaer Polytechnic Institute, USA Prior to that he graduated from University of Bradford with BEng (Hons) in Electronics, Computer and Communication Engineering. His research interests are in the area of VLSI Design and Testing, FPGA prototyping for Artificial Intelligence application and Encryption. He is currently pursuing his PhD at the University of Southampton, UK.

Ivan Teh Fu Sun is a design engineer at Intel technology, Penang, Malaysia. He obtained his BEng (1st Class Hons) from Universiti Teknologi PETRONAS, Malaysia.