IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 2, April-May, 2013
ISSN: 2320 - 8791
www.ijreat.org

Implementation of Feed Forward Neural Network Using Layer Multiplexing for Effective Resource Utilization in FPGA

R.Asha[1] P.Bowrna[2] and F.Mhaboobkhan[3]

Electronics and Communication Engineering (ECE).
MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI.

***Abstract*— This paper presents a hardware implementation of Neural Networks in reconfigurable field programmable gate array (FPGA). The need for more number of multipliers and adders in Neural Networks limits the size of network. In this paper we propose a technique to minimize the resource by implementing the layer multiplexing of the network. Therefore the resource utilized is much reduced without much reduction in the speed. Thereby we can implement a huge network in application specific integrated circuit (ASIC) at low cost. In this paper instead of implementing the complete network, we implement only the largest layer in the network. Then using the control block we can use that largest layer to function as different layers of network. The control block contains the inputs, weights, bias and excitation function for every layer of network. It ensures proper functioning by assigning appropriate information to the layer being computed. This concept is effective in reducing the resource requirement at the cost of a moderate overhead on speed. This makes the NN application viable to cost and speed for online applications. Multilayer networks have been implemented using Xilinx FPGA.**

***Index Terms*—Artificial Neural Network (ANN), FPGA(Field Programmable Gate Array),Layer Multiplexing, Neural Networks, Verilog.**

I. INTRODUCTION

THE emergence of artificial neural networks is the idea to build machines more like humans. The special feature of artificial neural network (ANN) is their ability to learn as they get trained. They have additional features like robustness to noise and fault tolerance which paves way for many applications to various fields. Real time applications are in need of high speed neural computations with minimum number of neural layers. So layer multiplexing helps to reduce the number of layers to a single large layer combining the functions of other layers. Implementation of neural network can either be by analog or digital hardware. Analog systems are more difficult to design and implement on the other hand digital systems are known for their higher accuracy, flexibility, repeatability and have better flexibility. The digital neural network implementations can be done using field programmable gate array (FPGA) as it performs parallel processing easier than other DSP and ASIC processors.

Consider a network with three layers comprising of an input layer, a hidden layer and an output layer. To realize all types of nonlinearity using three layers, large number of neurons is needed in the hidden layer resulting in a massive NN.

The size and complexity of an NN depends on both the total number of neurons and the number of layers. In the implementation of NN, parallel computation has the advantage of high speed but resource requirement is large which in turn increases the system cost. Sequential operation reduces resource but results in lower speed of operation. Hence, the requirement of high speed and low cost is addressed in this paper.

This paper proposes a simple architecture to implement a complete NN using minimum resource regardless of the size of the network. A single largest layer (i.e., layer with maximum neurons) is implemented instead of implementing the complete network. This layer calls itself repeatedly and behaves like the different layers of the network. The proposed multiplexed architecture implementation greatly reduces the resource requirement, thus making multilayer ANNs realizable with low-cost FPGAs.

II. STRUCTURE OF A SINGLE NEURON

A. Computational blocks of a single neuron:

The basic structure of a neuron with n inputs is shown below. Let $p_1,p_2\ldots\ldots p_n$ be 'n' inputs, $w_1,w_2,\ldots\ldots w_n$ be corresponding weights and 'b' be bias.

Let f(x) be the non linear excitation function. The processing done by a neuron is described by the following

$$y = f(x)$$

where

$$x = \sum_{i=1}^{n} p_i w_i + b$$

f(x), the non linear excitation function can be any excitation function such as linear, log sigmoid or tan sigmoid function.

1) Linear

$$f(x) = x$$

2) Log-sigmoid

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

3) Tan-sigmoid

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

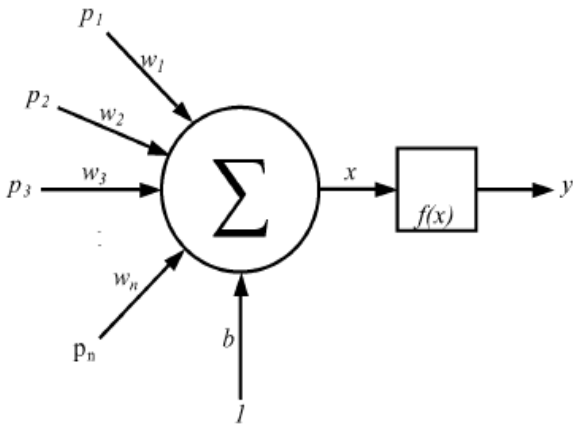

Figure 1: Structure of a neuron

The single-neuron implementation basically requires computational elements such as simple adder, multiplier, and complex evaluator of the nonlinear excitation function[2]. Hence, the computational blocks in a single-neuron implementation are the addition block, multiplication block, and excitation function block.

B. Precision of the Blocks in Single-Neuron

Precision is the important parameter in the digital representation of a neuron which decides the output resolution. Higher resolution needs large resource requirement which ultimately increases the cost[3].

The inputs are binary numbers which is either 0 or 1. The weights are signed numbers with 1 sign bit, 3 bits for whole part and 4 bits for fractional part. The bias is also signed numbers with 1 sign bit, 3 bits of whole part and 4 bits of fractional part.

C. Implementation of a Single Neuron in FPGA

The structure of a neuron is split into various computational sub-blocks and these blocks are implemented individually, and then they are integrated to form the complete neuron[1]. The various sub-blocks are detailed as follows:

1) MUL8: It performs multiplication of two signed numbers. The inputs are obtained from the main control block.
2) ADD: It performs addition of signed numbers. The inputs are obtained from main control block and the output is returned to the same control block.
3) SIGMA_CTRL: This block is the main block where all other sub-blocks are integrated together.

a) It places correct weights and bias for multiplication

b) It knows the number of layers and number of neurons in each layer.

c) After performing the corresponding multiplication and sum for that result we find the excitation function.

III. Layer Multiplexing in the network

The layers of a neural network consist of input layer, hidden layer and output layer. The hidden layers can be one or many with varying number of neurons. Figure 2 shows the general network with n layers.

S^0 shows the input layer, S^1 to S^{m-1} shows the hiden layers and S^m shows the output layer.

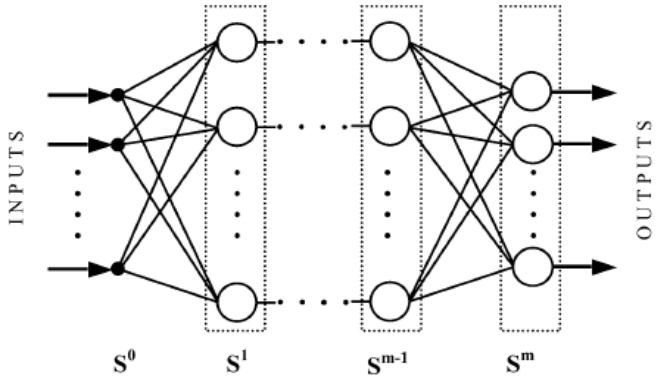

Figure 2: General network with n layers

The layer with maximum number of neurons is implemented and the remaining layers are accessed within the single largest layer by making the unused neuron's input as zero. This is layer multiplexing in neural network, shown in figure3. For example consider a network 2-3-5-8-1 i.e. with two input neurons and 1 output neuron with three hidden layers of 3, 5 and 8 number of neurons. The layer with 8 neurons is implemented and the remaining layers are multiplexed within that layer.

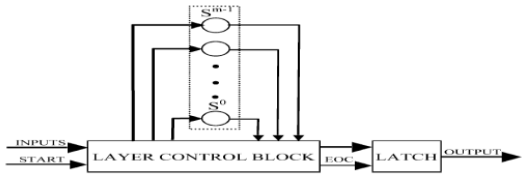

Figure 3: General layer multiplexing

A. Implementation of Layer-Multiplexed NN

Implementation details of a layer-multiplexed NN are presented in this section. Four different network architectures are such that for all the networks the single largest layer

contains five neurons and each neuron has eight inputs. The architecture are as follows:

1) 8-5-3;
2)8-5-5-3;
3)8-5-5-5-3;
4)8-5-5-5-5-3;

All those NNs have eight inputs and three outputs. The log sigmoidal functional is used for the hidden layers and output layer. This choice of network is to illustrate the saving in resource, which increases as the number of layers increases. So we implement only the largest layer of NN along with a control block. The control block contains information about the number of layers and number of neurons in each layer. It also contains the weights and bias weight of each layer. So the control block will have knowledge about the layer it is processing. So it picks the corresponding weights and bias to compute the output of each layer. The output x is then converted to f(x) using the log-sigmoidal excitation function for the hidden and output layer alone[1].

The sequential operation of the control block is detailed as follows:

1) All the weights and bias of the neurons get initialized.
2) The corresponding input gets multiplied with weight and bias is added. The output x obtained at this stage is converted to f(x) using log-sigmoidal excitation function.
3) This output is then fed to the next layer. Since we are going to implement only single layer, we fed the output obtained in this layer back to the implemented single layer.
4) If we are not going to use all neurons in the implemented largest layer then we make the inputs for those neurons as Zero.
5) The process repeats until we meet the final output result.

The various steps involved in FPGA design flow are as follows: 1) design entry, 2) synthesis, 3) simulation, 4)implementation, and 5) device programming. The code is simulated in ModelSim ALTERA STARTER EDITION 6.4a and executed in Xilinx device.

This technique of Layer Multiplexing can reduce the resource requirements and improve the speed. It can save resource upto 50% and speed is improved upto 17%. As the operating clock frequency of FPGA is high, the extra overhead in cycles does not affect overall operation of the network.

IV. FPGA implementation of Neural based Xor function

This paper proposes a implementation of NNs using FPGA with minimum resource without much loss in speed. The resource reduction technique is demonstrated by implementing a neural-based XOR function using FPGA. Here NN based XOR function 3-5-1 network is chosen. Here we implement only the single largest layer with 5 neurons. Log-sigmoid function is used for the hidden layers and the output layer of the network. To determine the weights and bias, we trained the network in Matlab[5]. By implementing this technique we require only five neurons. Otherwise it would require 9 neurons to implement the complete network. Higher Bit precision would increase the accuracy but would increase the resource requirements also. However, for the given bit precision the proposed technique would minimize the resource requirement and would cost less without much compromise in the speed.

V. CONCLUSION

Multilayer feed forward NNs have been implemented using FPGA. The implementation of only largest layer along with a control block would minimize the resource. The control block ensures proper functioning by assigning simultaneously appropriate weights, biases, and excitation function. Different network configurations have been implemented without and with layer multiplexing. The saving in resource would increase with the increase in the number of hidden layers. With increase in the hidden layers, the control block complexity would increase resulting in slight overhead on speed. The large saving in resource leads to reduced cost making NN applications more feasible and promising.

REFERENCES

[1] S.Himavathi,D.Anitha, A.Muthuramalingam "Feedforward Neural Network Implementation in FPGA using Layer Multiplexing technique" in IEEE Transactions on Neural Networks, Vol. 18, No. 3,May 2007
[2]S. L. Pinjare, Arun Kumar M, "Implementation of Neural Network Back Propagation training Algorithm on FPGA" in International Journal of Computer Applications(0975-8887), Volume 52-No. 6,August 2012
[3] A.Muthuramalingam, S.Himavathi, E.Srinivasan "Neural Network Implementation using FPGA: Issues and Application" in International Journal of Information Technology Volume 4 Number 2
[4]X. Yu and D.Deut, "Implementing neural networks in FPGA" in Proc. Hardware Implementation Neural Network Fuzzy Logic Inst. Elect. Eng. Colloq.,Mar. 9,1994,pp. 326-329
[5]K. M. Hornick, M. Stinchcombe, and H. White, "Multilayer feedforward neural networks are universal approximators," Neural Netw., Vol. 2, No. 5,pp. 141-154,1985.
[6] Naleih M. Botros and M. Abdul-Aziz "Hardware Implementation of an Artificial Neural Network Using Field Programmable Gate Arrays (FPGA's)" ieee transactions on industrial electronics VOL. 41. NO 6. December 1994.