A Generic Control Block for Feedforward Neural Network with On-Chip Delta Rule Learning Algorithm

Alin Tisan¹, A. Buchman¹, Ş. Oniga¹ and C. Gavrincea¹
¹) Electrotechnical Department, North University of Baia Mare, Baia Mare, Romania
atisan@ubm.ro

Abstract: In this paper we propose a method to implement in FPGA a feedforward neural network with on-chip delta rule learning algorithm. For this, we have develop a generic blocks designed in Mathworks’ Simulink environment, capable to generate the signals for controlling the neurons from a neural network. The main characteristics of those blocks is its high reconfigurability that’s makes it suitable for developing of a generic controlling block capable to manage calculus function of neurons from different layers. The properties of the block are set according to the numbers of total layers, number of the neurons from the layers and the number of layer from whom and in different Function Block Parameters windows. The novelty of the proposed method resides in the possibility to design neural networks with on-chip learning only with predefined block systems created in System Generator environment. The major benefit of this design methodology result from the possibility to create a higher level design tools used to implement neural networks in logical circuits.

1. INTRODUCTION

FPGA implementation of a neural network is a mandatory issue for developing of an intelligent digital system that is able to learn and auto reconfigures in order to process data in a proper manner [1]. System Generator is a toolbox from Mathworks’ Simulink environment that provides basic Xilinx Blockset used for designing of a system able to computes according to a neural network algorithm.

The entire proposed concept relies on the idea that a FPGAs implementable neural network can be reached only by choosing the predesigned generic blocks and to set the parameters of the network into a pop-up window.

In order to design these generic blocks, used for neural network building, it was developed an algorithm for setting the customizable block parameters according to the chosen network topology.

From point of view of the role of the blocks into design, these are divided in control blocks and computing blocks.

The control blocks are designed in MCode and VHDL code and incorporated into system by Black Box HDL and MCode Blocks. The role of these blocks is to manage the control signal of the processing bloc in order to initialize and command the processing components.

Some of the calculus blocks emulate the McCulloch - Pitts neuron model behavior and some of them will manage the learning phase. Thus, the major feature of these is to implement those functions that calculate the neural output and the weights concordantly with the learning rule adopted and the neural network topology, such as number of layers, number of neuron from a layer and data representation.

The chosen network is a feedforward one and the corresponding weights are calculated with Delta rule. In this way the formulas that have to compute by the calculus blocks are divided in two categories: the formulas involved in propagation phase and the formulas involved in the learning phase.
2. CONTROL ALGORITHM FOR PROPAGATION PHASE

In the propagation phase the output of the neural network is given by the following equation:

\[
output = \frac{1}{1+e^{-(b+\sum_{k=1}^{N} w_k x_k)}}
\]

(1)

Where \(b\) is the bias value, \(x_k\) is input and \(w_k\) is corresponding weight.

The adopted neuron model consists of blocks that compute the neural layer output: a multiplier, accumulator, RAM and the multiplexer (fig. 1).

In order to manage the entire network, the control block will generate a generic time base according with the chosen neural network architecture, i.e. number of layers, neurons per layer, fig 3.

Fig. 1. Hardware components of a neuron

The neurons are controlled by a generic block driven by two signals, fig 2.

Fig. 2. Generic control block

The propagation/learning signal will set the neural network behavior in a propagation phase (the weights are set and network will recognize the input pattern) or in a learning phase (the weights must be calculated and set).

The second signal will start the computing process for a given situation: learning or propagation.

Fig. 3: Function Block Parameters windows for General counter block

Also, for a given architecture, three layers in this case, the control block will generate the values of the signals needed to control the computing elements of a neuron, fig. 4.

Fig. 4: Architecture of the control block

The General Counter block described in VHDL code and incorporated into design by a black box HDL, fig. 5

Fig. 5: VHD General Counter implementation
The signal generator blocks consist of two blocks designed in MCode, Subsisem1 block and VHDL, Black box.

![Fig. 6: Architecture of the Signal Generator block](image)

The Subsisem1 block will provide the constant values used by Black Box. These constants are calculated based on properties of a layer (hidden or output), which are applied to, fig 7.

![Fig. 7: MCode for constants calculation](image)

where $t$ is the number of layer and $n$ is the number of the neurons from the $t$ layer.

The other block, designed in VHDL code uses the constants generated by the Subsisem1 and set the signal to activate the calculus elements of the neuron.

The most important part of the program described in VHDL is given by the process statement that compares the value of General counter with the constants provided by Subsisem1 and set or reset the control signals, fig. 8.

![Fig. 8: VHDL Code for signal settings](image)

3. CONTROL ALGORITHM FOR LEARNING PHASE

In the learning phase the control block has to manage the implementation of the Delta rule used to calculate the weights that minimize the total error.

Thus, the weight of the neurons from the out layer will be calculated with the formula (2)

$$w_{kj}^{\text{new}} = w_{kj}^{\text{old}} + \eta (d_k - o_k) o_k (1 - o_k) y_j$$

(2)

where $w_{kj}^{\text{old}}$ and $w_{kj}^{\text{new}}$ are the weights before and respectively after the calculus, $\eta$ is learning rate, $o$ is the output value and $d$ is the target value.
In the case of the hidden layer, layer “n-1”, (where the “n” layer is the output layer) the new weights, $v_{ij}$ are calculated according to the formula (3):

$$v^{(n-1)}_{new} = v^{(n-1)}_{old} + \eta f(n_{ij})^{n-1}(1-f(n_{ij}))\cdot f(n_{ij})^n \sum_k \delta_k w_{jk}$$

(3)

In this way, using (3), we can calculate the weights of the neurons from all hidden layers, [1].

The above formulas are hardware implemented with the aid of the Neural Network Xilinx BlockSet User Library [2, 3], fig 9.

Beside the logic blocks, the hardware architecture of the neural network has to include a series of blocks that calculate the proper delay of the signals. In the calculus process, due to the node parallelism of the neurons, the data are parallelized. For this, the data has to be delayed with a value proportional with the number of the neurons from the hidden layer and the number of the layer according with the following formula:

$$Delay = n_2 \cdot ceil(((n_1+n_2)+t\cdot6+12)/n_2) - (n_1+n_2)+t\cdot6+15$$

(3)

where $n_1$ and $n_2$ are the input and hidden neurons and “t” is the number of layers.

The delay blocks are implemented using configurable Xilinx Delay blocks and the delaying value is set by corresponding pop-up windows, fig 9.

4. CONCLUSION

The proposed model is constituted by neuronal blocks, organized in hidden and output layers, the control blocks that control and command the behavior of the neurons elements and the calculus blocks that compute the weights and the outputs of the network.

With the new created library from blocks designed in Simulink environment, the designer will be able to develop the entire neural network by parameterize the ANN topologies as number of neurons and layers.

The implementation goal is achieved using the Mathworks’ Simulink environment for functional specification and System Generation to generate the VHDL code according to the characteristics of the chosen FPGA device.

REFERENCES

