FPGA Based Hardware Implementation of a Self-Organizing Map

S.T. Brassai*
* Sapientia Hungarian University of Transylvania/Department of Electrical Engineering, Tîrgu Mureş, Romania
e-mail tiha@ms.sapientia.ro

Abstract—An embedded parallel pipeline solution for hardware implementation of Self Organizing is discussed. The Kohonen Self Organizing map was successfully applied in the travelling salesman problem for a robotic mobile agent application. The theoretical background of the application was discussed in a pre-study [1], current paper focusing on hardware implementation of the self-organizing map. The strength of the solution presented in the paper results from the parallel-pipeline architecture and the parallel computation of the output and the weight update. On hardware implemented processing units the SOM neurons are processed sequentially. Solution for parallel processing of the network output and weight update based on use of dual port BRAM memory, which enables to read and modify the values of the weights in same clock cycle, is presented. The number of hardware neurons used depends on the resources of the used FPGA. From the hardware implemented neural network an IP core was generated and integrated to a Microblaze processor bus system, enabling the programming of the system parameters and testing the system in real applications.

I. INTRODUCTION

The artificial neural networks’ structure allows highly parallel hardware model implementation in FPGA circuits. Thus, to the growing use in embedded systems of hardware implemented artificial neural networks, providing real-time operation of the application meeting with many of these hardware implementations of artificial neural networks like Multilayer Perceptron (MLP), Radial Basis Function (RBF) and Cerebellar Model Articulation Controller (CMAC), a number of hardware implementation of self-organizing map could be encountered.

The increasing capacity and higher operating frequency of FPGA circuits with relative low power consumption motivate the implementation of the highly parallelizable and high complexity systems in FPGA circuits. In this paper an efficient hardware implementation in FPGA circuit of the self-organizing map with parallel-pipeline architecture is presented.

In accordance with the use of the self-organizing map in a TSP application [1] an artificial neural network structure with two inputs was considered to be used, but the proposed architecture can easily be converted to a system with more than two inputs. In this case the circuit is limited by the capacity of the FPGA circuit and two inputs were applied.

The full parallel architecture has a high-demand of hardware resources, so an intermediate solution was chosen: on physically implemented parallel neurons multiple neurons are processed sequentially [2][3].

From the point of view of hardware implementation in FPGA circuits of artificial neural networks some considerable aspects are presented:
- number of bits used for data representation (network inputs, weights, activation functions)
- arithmetic, which is used to represent the data (fixed-point, floating point)
- how much BlockRAM memory is in the system?
- how many parallel multiplier or DSP modules are in the FPGA circuit?

The number of configurable logic blocks in the FPGA system and, of course, the FPGA circuit family as for the complexity of the slice’s structure can be simple or more complex.

From experience gained from the variety of large complex applications implementation in FPGA circuits (artificial neural networks, continuous wavelet convolution discrete implementation, hardware fuzzy controller) and the studied literature based implementations as well as of the capacity of the FPGA development system, various parameters encoding was pre-estimated and applied to the architecture for the hardware implemented Kohonen network.

In the designed and implemented Kohonen network discussed in this paper a fixed-point arithmetic is used. With minimal modifications, the network structure can be converted to floating-point version. Simply, instead of the fixed-point operations floating-point IP cores can be used. In a high capacity FPGA circuit the floating-point arithmetic can easily be applied.

In the next section, starting from the structure of the hardware neurons, the hardware structure of the artificial neural network is realized and then the structure of the system used for testing is discussed. For testing, the hardware implemented self-organizing map was connected to the peripheral local bus of the used Microblaze core-based processor.

II. IMPLEMENTED SELF-ORGANIZING MAP

A. Architecture of hardware implemented self-organizing map

The proposed Kohonen network’s logical structure is illustrated in the following figure 1. The neurons from the artificial neural network are evenly distributed between
physical neurons and are called in the following logic neurons.

A processing unit (neuron) basically is composed from two subunits: the neuron itself and the winner neuron selection modules. A physical neuron with a pipeline-parallel architecture is used to process sequentially multiple logical neurons and is composed from the following subunits: memory for weights storage, logic to calculate the neuron output, and the teaching modules.

One of the basic advantages and characteristics of the proposed architecture, which has been successfully applied to the RBF network hardware implementation in FPGA circuit, is the use of the dual port BRAM (Block RAM) memory.

One of the two ports is used to calculate the logic neuron output (and programming and reading of the weight values), and the other to store the newly calculated weight values. In the current version, 16 bits were used for the weights and the inputs encoding.

A 16kbit BRAM memory configured with 512 address locations and 32 bit data bus can store in an address location one set of weights of a logic neuron associated with each input of the network. As also detailed in the following the same configuration of BRAM memory is used for storing the training set.

The hardware implemented neural network logic structure is illustrated in the figure 1. The winning neuron selection module inputs are connected to the current neuron outputs and the outputs of the left and right neighboring neurons.

B. Structure of the output processing module

Because of Euclidian distance implies the calculation of the square root, instead of the Euclidean distance a simpler norm such as the Manhattan norm or maximum norm is proposed for distance operation since the hardware implemented square root operation requires a high number of hardware resources. With the use of a proper high capacity FPGA circuit for square root operation the CORDIC LogiCORE ™ IP core can be used.

The neuron output is calculated according to the following equation

\[
y_{i,p} = \max \left\{ \sum_{j=1}^{M} (w_{i,p,j} - x_{q,j}) \right\}
\]

where \( y_{i,p} \) represents the output of the neural network for the i-th sequential neuron processed on p-th physical respectively \( \overline{w}_{i,p} \) the weight vector of the i-th logical neuron of the p-th parallel processing unit, the q-th input vector from the training set, M represents the number of elements of the input vector. In general the neuron output can be written in the following form:

\[
y_{i,p} = \max \left\{ \sum_{j=1}^{M} (w_{i,p,j} - x_{q,j}) \right\}
\]
presented structure new registers is needed in the index delay channel.

As it was discussed in the first part of the paper, the TSP task can be solved with using the Manhattan and maximum norms.

C. Winner neuron selection module

In the present case for the winner selection the criteria is the minimum distance between to the input vector and the weight vectors. The winner neuron selection is made in two units as in the following figures is presented:

![Figure 3. Winner selection module implementation block diagram](image)

In the first step, the minimum value is selected from the sequentially processed neuron output, the minimum value propagated from the second level module output, and the current minimum value which is stored in each step in the module, respectively in the second level module from the current minimum value, the output of the first unit for current neuron and outputs of the first level unit for neighbor neurons.

The index of the winner neuron consists of two of the sequential processed neuron indexes respectively the parallel processing unit number: \( I = 2N^*p + j \) where \( N \) is the number of neurons processed sequentially on physical neuron and \( j \) the-sequential index of current processed neuron on \( p \)-th parallel processing unit.

In principle the minimum value selection implemented in two separate modules can be combined into a single module, but in this case the pipeline module will operate on a much smaller clock frequency (the clock frequency is reduced from approximately 150 MHz to about 90 MHz as resulted after model synthesis)

\[
[m^*,I^*] = \min_{p=1..P,i=1..N}(v_{p,i}^*,\tilde{x}_q^*)
\]  

\[
\Phi(i_q^*,I^*_{q-1}) = \begin{cases} 
0 & \text{if } |I_q - I_{q-1}^*| > md \\
md - |I_q - I_{q-1}^*| & \text{if } |I_q - I_{q-1}^*| \leq md 
\end{cases}
\]

where \( I_q^*, I_{q-1}^* \) index of winner neuron and index of currently trained neuron, \( md \) a configuration parameter, \( \tilde{x}_q^* \) q-th input vector, \( \Phi(i_q^*,I_{q-1}^*) \) neighborhood function.

D. Weight update module.

The weight update is realized in parallel with the network output processing with one step-delay. When the network output is processed for the q-th input vector \( \tilde{x}_q \) the weight update is processed for \( \tilde{x}_{q-1} \). The new weight value is obtained according to the normalized Hebb rule presented in the equation:

\[
\tilde{w}_{i,p}[k + 1] = \tilde{w}_{i,p}[k] + \mu \Phi(i_q^*,I_{q-1}^*) (\tilde{x}_{q-1} - \tilde{w}_{i,p}[k]).
\]  

\[
\mu = \frac{1}{2^o}.
\]

The learning parameter has the form presented in (6), where \( o = 1..15 \).

The new values of the weights are processed in six phases in the weight update module based on parallel pipeline architecture. For each phase the operations are executed in parallel for both weights.

In the first step, the weights are read out from the BRAM memory. This is essentially the result of calculation used by the output processing module.

During the first step the absolute value of difference between the neuron index value and the winner neuron index value \( |I_q - I_{q-1}^*| \) will be calculated as required for calculation of the neighborhood function.

In the second step, according to the normalized Hebb rule, the \( \{x_{q-1} - w_{i,p,l}[k]\} \) and \( \{x_{q-1} - w_{i,p,2}[k]\} \) operations are carried out. The neighborhood function value \( \Phi(i_q^*,I_{q-1}^*) \) is also needed to be calculated in the second step.

![Figure 4. Weight update module block diagram](image)

In the next step, using the partial results \( \Phi(I_q^*,I_{q-1}^*) \) respectively \( \{x_{q-1} - \tilde{w}_{i,p}[k]\} \) received in the previous step the multiplying operations is performed for both weights of the logical neuron with index \( i \) from \( p \)-th parallel unit.

Since fixed-point arithmetic was used the teaching coefficient is encoded in the fixed-point complement of the two representation forms. The teaching coefficient was coded according to formula (6), this will ensure that the teaching factor has value in the \([0...1]\) interval and instead of the multiplication operation a division operation of powers of two is applied.
The divide operation is highly hardware consuming, but the division with powers of two can be simply performed by shifting the bit sequence to the right. The implementation of this operation is carried out in the 4th step.

The learning coefficient and the max distance (md) value used in the neighborhood function are changing during the teaching process, the used values are critical for a successful training task. For both parameters the values change can be made internally in the controller assigned to the network, or the values can be accessed from an external register programmed by the control processor. One of the bits of the control register can be selected to be used as internal or external values of the md or learning coefficient.

The next step is shown in the following figure, and it is the calculation of the new weights values and a summing function is performed.

The values of weights are available after the reading phase, at the time of the summing operation on the output of the BRAM memory the x-th neuron weights have already been read. The weights are transferred trough a three-level delay channel implemented with registers, thereby ensuring that the weights are ready for the summing operation at the right moment.

In the last phase the newly calculated weight values written back into the dual port BRAM memory of port B are performed. Between the weight’s read and write-back phase for the same neuron there is a phase delay of 5 steps. The memory address which is used to read out the weights in the current clock cycle j cannot be used to store the new weights for the neuron j−5. The memory addresses are passing through a five-level delay channel.

After the three-level delay channel used in neuron output processing module, a two-level delay is used, called the address delay register.

The use of dual-port BRAM memory is very important from the point of view of the teaching algorithm implementation. The dual-port memory allows the read and write operation at the same time.

For synchronization of different parts of the implemented neural network (neuron output processing modules, the training module, the winner neuron selection unit) a controller implemented as a 6-state finite state machine is responsible for. The control unit is configurable on two input registers, and the controller states with debugging role can be read-out from a state register.

Using the first input register the system can be reset, running the output processing, changing between the programming and running mode, enabling or disabling the learning, selecting the external or internal learning coefficient and md to be used.

Also in the first register the learning coefficient is defined and for the maximum distance an external learning coefficient or max distance is chosen.

The second register bits are used to select which neuron or input BRAM memory is considered to read or write. The block diagram of the implemented self-organizing map and the connection to the Microblaze processor PLB bus is presented in the following figure, where a system with four hardware implemented processing units was configured.

III. CONCLUSIONS

The modules of the discussed self-organizing map were implemented in VHDL. The resulted solution is very fast, the neuron outputs are processed in three clock cycles, the weight update in six clock cycles.

Due to the fact that the output processing and the weight processing is realized in the same time, respectively due to the parallel pipeline architecture the FPGA presented implementation is incomparably faster than a CPU-based sequential solution. For an input vector the network output processing and the weight update is realized in a number of clock cycles approximately equal with the number of neurons processed on a physical hardware. Doubling the hardware neurons in the system the processing time of the network output is reduced by half.

Structure and capacity of the available FPGA development systems allow the implementation of increasingly large neural networks. Models of embedded neural networks proposed and tested in the previous years on low-capacity FPGA circuits could be efficiently implemented and applied in real applications.

Figure 5. Block diagram of the implemented self-organizing map and the connection to the Microblaze processor PLB bus

REFERENCES


