Hindawi Publishing Corporation
Advances in Artificial Neural Systems
Volume 2014, Article ID 602325, 10 pages
http://dx.doi.org/10.1155/2014/602325

Research Article

Architecture Analysis of an FPGA-Based Hopfield Neural Network

Miguel Angelo de Abreu de Sousa, Edson Lemos Horta, Sergio Takeo Kofuji, and Emilio Del-Moral-Hernandez

University of São Paulo, 05508-010 São Paulo, SP, Brazil

Correspondence should be addressed to Miguel Angelo de Abreu de Sousa; miguel.angelo@usp.br

Received 30 June 2014; Revised 11 November 2014; Accepted 12 November 2014; Published 9 December 2014

Academic Editor: Ping Feng Pai

Interconnections between electronic circuits and neural computation have been a strongly researched topic in the machine learning field in order to approach several practical requirements, including decreasing training and operation times in high performance applications and reducing cost, size, and energy consumption for autonomous or embedded developments. Field programmable gate array (FPGA) hardware shows some inherent features typically associated with neural networks, such as, parallel processing, modular executions, and dynamic adaptation, and works on different types of FPGA-based neural networks were presented in recent years. This paper aims to address different aspects of architectural characteristics analysis on a Hopfield Neural Network implemented in FPGA, such as maximum operating frequency and chip-area occupancy according to the network capacity. Also, the FPGA implementation methodology, which does not employ multipliers in the architecture developed for the Hopfield neural model, is presented, in detail.

1. Introduction

For nearly 50 years, artificial neural networks (ANNs) have been applied to a wide variety of problems in engineering and scientific fields, such as, function approximation, systems control, pattern recognition, and pattern retrieval [1, 2]. Most of those applications were designed using software simulations of the networks, but, recently, some studies were developed in order to extend the computational simulations by directly implementing ANNs in hardware [3].

Although there were some works reporting network implementations in analog circuits [4] and in optical devices [5], most of the researches in ANNs hardware implementations were developed using digital technologies. General-purpose processors and application-specific integrated circuits (ASICs) are the two technologies usually employed in such developments. While general-purpose processors are often chosen due to economic motivations, ASIC implementations provide an adequate solution to execute parallel architectures of neural networks [6].

In the last decade, however, FPGA-based neurocomputers have become a topic of strong interest due to the larger capabilities and lower costs of reprogrammable logic devices [7]. Other relevant reasons to choose FPGA, reported in the literature, include high performance requirement which is obtained with parallel processing on hardware systems when compared to sequential processing in software implementations [8], reduction of power consumption in robotics or general embedded applications [9], and the maintenance of the flexibility of software simulations while prototyping. In this particular feature, FPGA presents advantages over ASIC neurocomputers because of the decreased hardware cost and circuit development period.

Among several studies on different network models implemented on electronic circuits, just recently were published in the literature works on the hardware implementations addressing specifically the Hopfield Neural Networks (HNNs). Those works usually aimed to approach the resolution of a target problem, such as, image pattern recognition [10], identification of DNA motifs [11], or video compression

[12] and only few studies concentrated on the peculiar features about FPGA implementations of HNN [13], which, unlike others neural architectures, is fully connected.

Driven by such motivation, the present work does the analysis of the chip occupied area according to the quantity of HNN's nodes and the precision required to represent its internal parameters. The paper is organized as follows. Next section briefly reviews the concepts of the network proposed by Hopfield, Section 3 describes details of the implementation strategy, and the last two sections present the results, conclusions, and discussions raised by this work.

2. Hopfield Neural Network

The HNN is a single-layer network with output feedback and dynamic behavior, in which the weight matrix W, connecting all its N neurons, can be calculated by

$$W = \sum_{k=1}^{M} \left(z^k \cdot \left(z^k \right)^{\mathrm{T}} \right) - M \cdot I \tag{1}$$

with each one of the M stored patterns represented by a vector z^k of N elements $z_i^k \in \{-1; +1\}$ and I denoting the identity matrix.

It has been shown [14] that, for a weight matrix with symmetric connections, that is, $w_{ji} = w_{ij}$, and asynchronous update of the neural activation, V_i, the motion equation for the neuron's activation always leads to a convergence to stable states. In the discrete-time version of the network, the neural activation is updated according to

$$V_i(t+1) = g(U_i) = g\left(\sum_{j=1, j \neq i}^{N} W_{ji} V_j(t) \right), \tag{2}$$

where $g(\cdot)$ is a sign function of the neural potential U_i and $V_i \in \{-1; +1\}$.

The model, proposed by Hopfield in 1982 [15], can also be interpreted as a content-addressable memory (CAM) due to its ability to restore prior-memorized patterns from initial inputs corresponding to noisy and/or partial versions of such stored patterns. The CAM storage capacity M increases according to N, and, with allowance for a small fraction of errors; references [16, 17] demonstrated the following relation:

$$M = 0.14N. \tag{3}$$

2.1. Application-Dependent Parameters. The number of nodes N of the Hopfield architecture is defined by the length of the binary strings stored by the associative memory, which is specific to each application. For example, when using the associative architecture for the storage of binary images with N pixels, the number of neural nodes has to be N [14].

With respect to M, which is the number of stored patterns in the associative Hopfield architecture, each one with length N bits, it is also defined by the requirements of the application: M is the number of distinct patterns relevant for the associative process. In the case of storage of images of decimal digits, for example, M will be 10. Notice though that the theoretical study of the Hopfield architectures indicates that we need to have a reasonable relationship between the number of stored patterns M and the size of the architecture N. According to [16], when M increases beyond $0.14N$ it results in significant degradation of the associative memory performance, and we say that the storage capacity of the architecture is exceeded.

3. Implementation Architecture

The FPGA-based HNN's digital system developed is depicted in Figure 1 and consists of a control unit (CU) and a data flow (DF) which implements the processes shown in the block diagram of Figure 2.

3.1. Data Flow. The input unit of Figure 2 receives the prompting input patterns and allows the update of neural activation V_i, according to the architecture illustrated by Figure 3, comprised of registers (R) and multiplexers (MUX).

The weight unit (Figure 2) is designed to execute the computation of $W_{ji}V_i$ without employing multipliers. The process is executed by registering the values of $+W_{ji}$ and $-W_{ji}$ and selecting between them based on the V_i state (−1 or +1). Figure 4 depicts the partial architecture of the weight unit, focusing on the implementation of the calculus involved in a specific neuron (neuron #1).

From Figure 4, it can be noticed that the proposed system implements the asynchronous version of the HNN, since the weight unit is designed in order to compute one weighted value (WY_i) at a time.

The summation unit, in Figure 2, performs the function $g(U_i)$ by inverting the signal bit resulting from the sum of the weighted neural activations V_i and feeding it back to the input unit. The architecture of this unit is shown in Figure 5.

The architecture of the final block in Figure 2, the output unit, is depicted in Figure 6. The unit outputs the state of activation for the entire network if a stable state is reached; that is, $V_i(t) = V_i(t+1)$ for $i = \{1, 2, \ldots, N\}$. The termination condition is identified by a comparison between the network state at the beginning and the network state at the end of an operational epoch, that is, after updating all neurons.

Finally, Figure 7 presents the circuitry used to implement the computations concerning one neuron. The illustration covers a single neuron (neuron #1) and depicts the interconnections between input, weight, and summation units. Connections to output unit are not shown.

3.2. Control Unit. The control unit is designed to activate the sequence of registers in data flow (Figure 1). More specifically, after the capture-input is triggered, the signals originated by control unit enable R_{IN} and R_Y in input unit, R_{WY} in weight unit, and R_{ADDER} in summation unit, sequentially. Control unit also generates the selection signals for the multiplexers in input unit to admit external data entrance only in the first cycle of operation and, in weight unit, in order to enable the computation of each neuron. At the end of an operational cycle, control signals are generated by the unit for

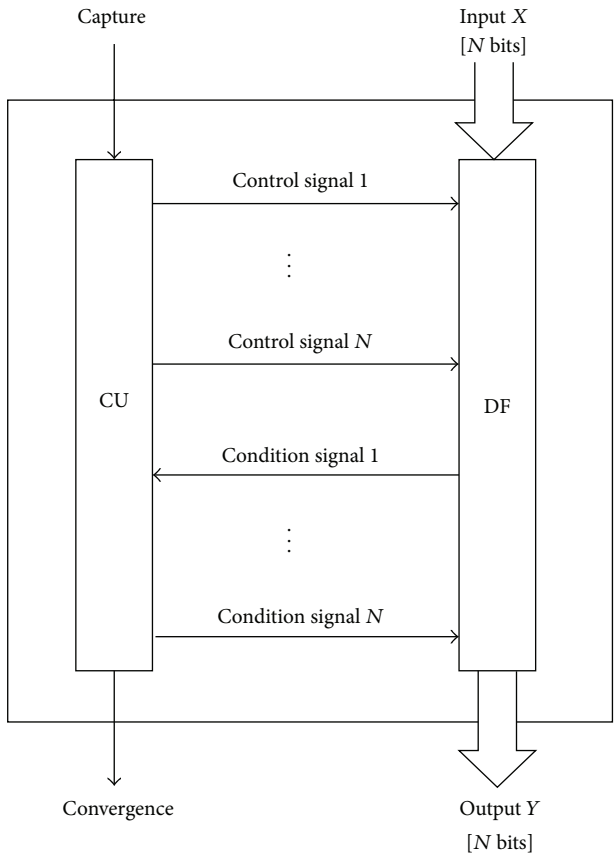

Figure 1: FPGA-based HNN's digital system.

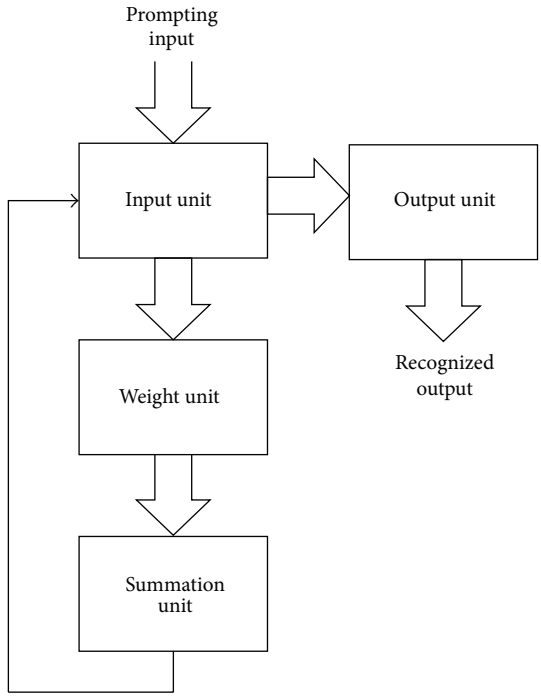

Figure 2: Data flow block diagram.

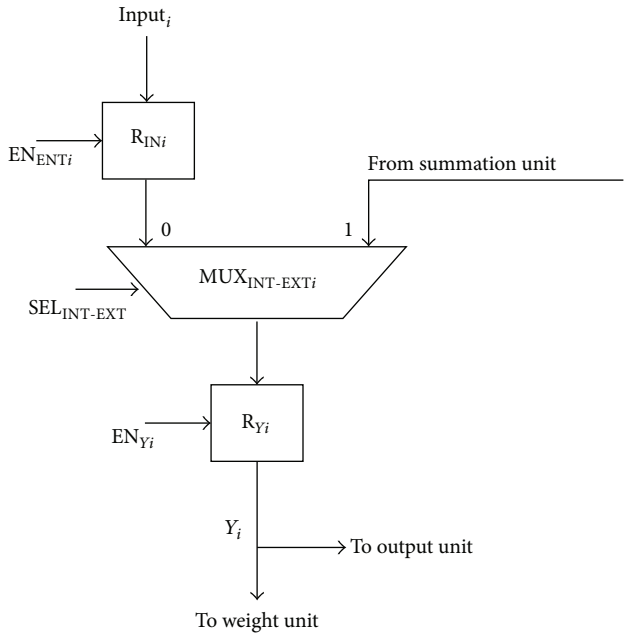

Figure 3: Input unit.

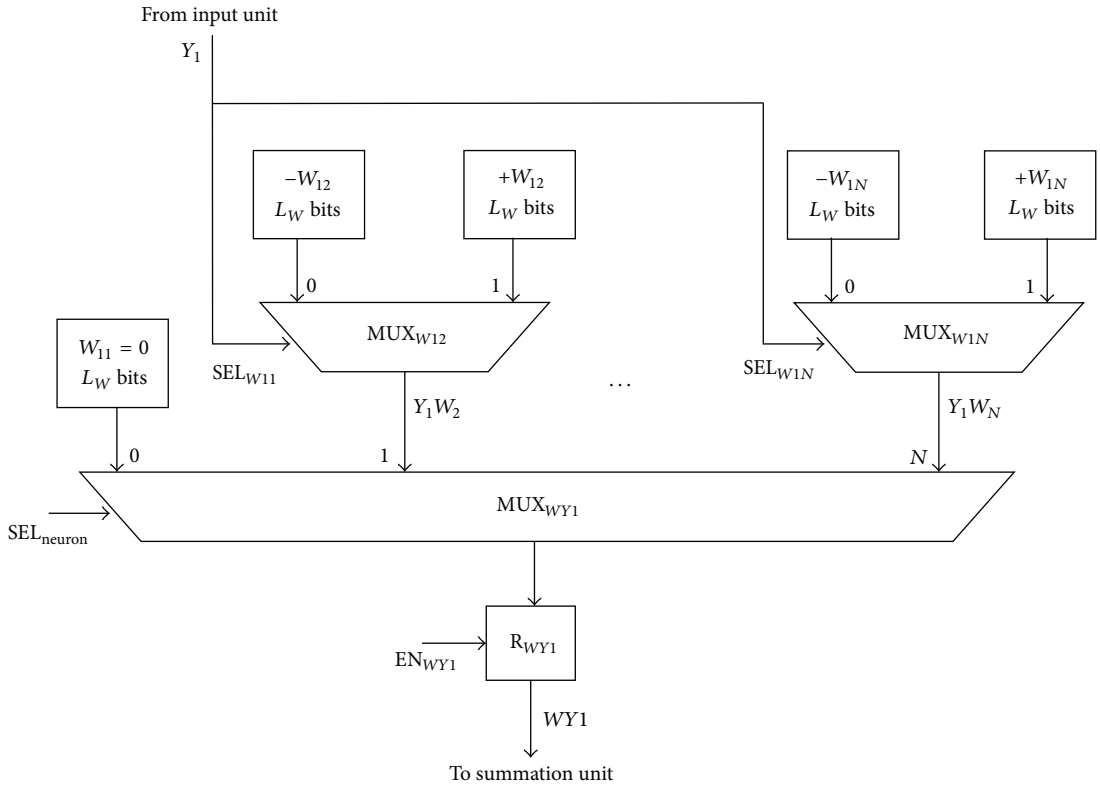

Figure 4: Weight unit.

the purpose of allowing comparisons between initial and final neural activations patterns of the entire network, in order to detect convergence. Algorithm 1 summarizes control unit by presenting the pseudocode of the state machine.

3.3. Dimensions of the Registers. Internal parameters of the network use two's complement representation and, due to the high number of interconnections in HNNs, it is important to establish the size of weight registers. From (1), the maximum

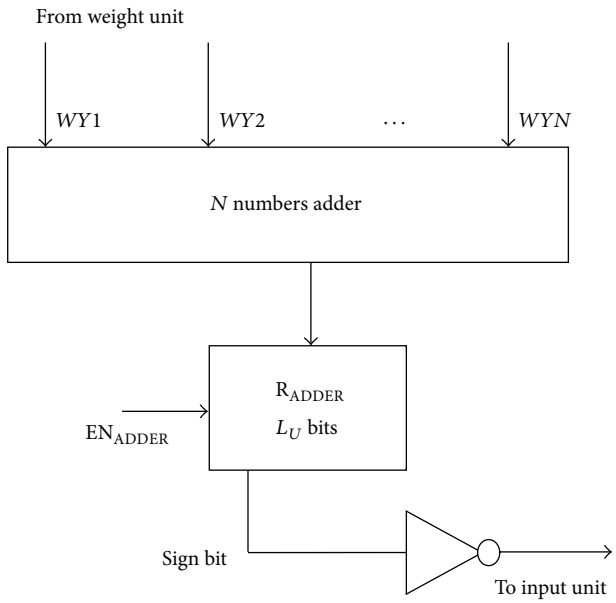

Figure 5: Summation unit.

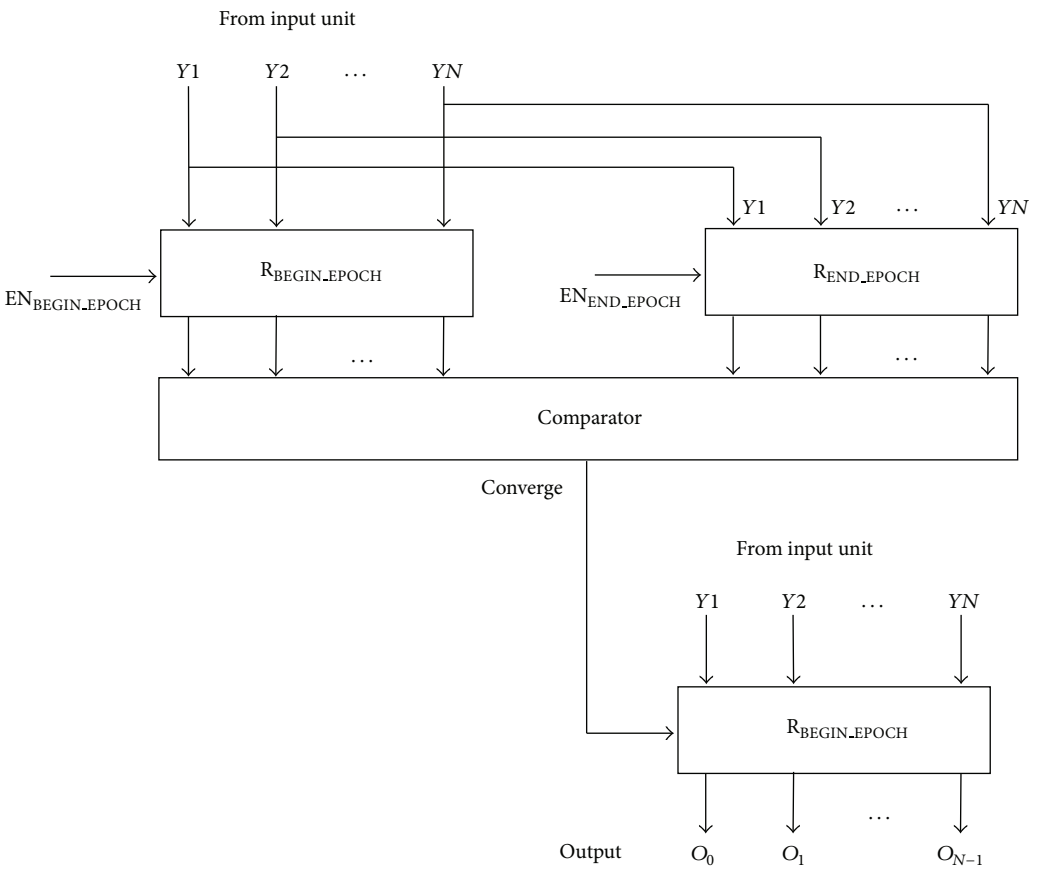

Figure 6: Output unit.

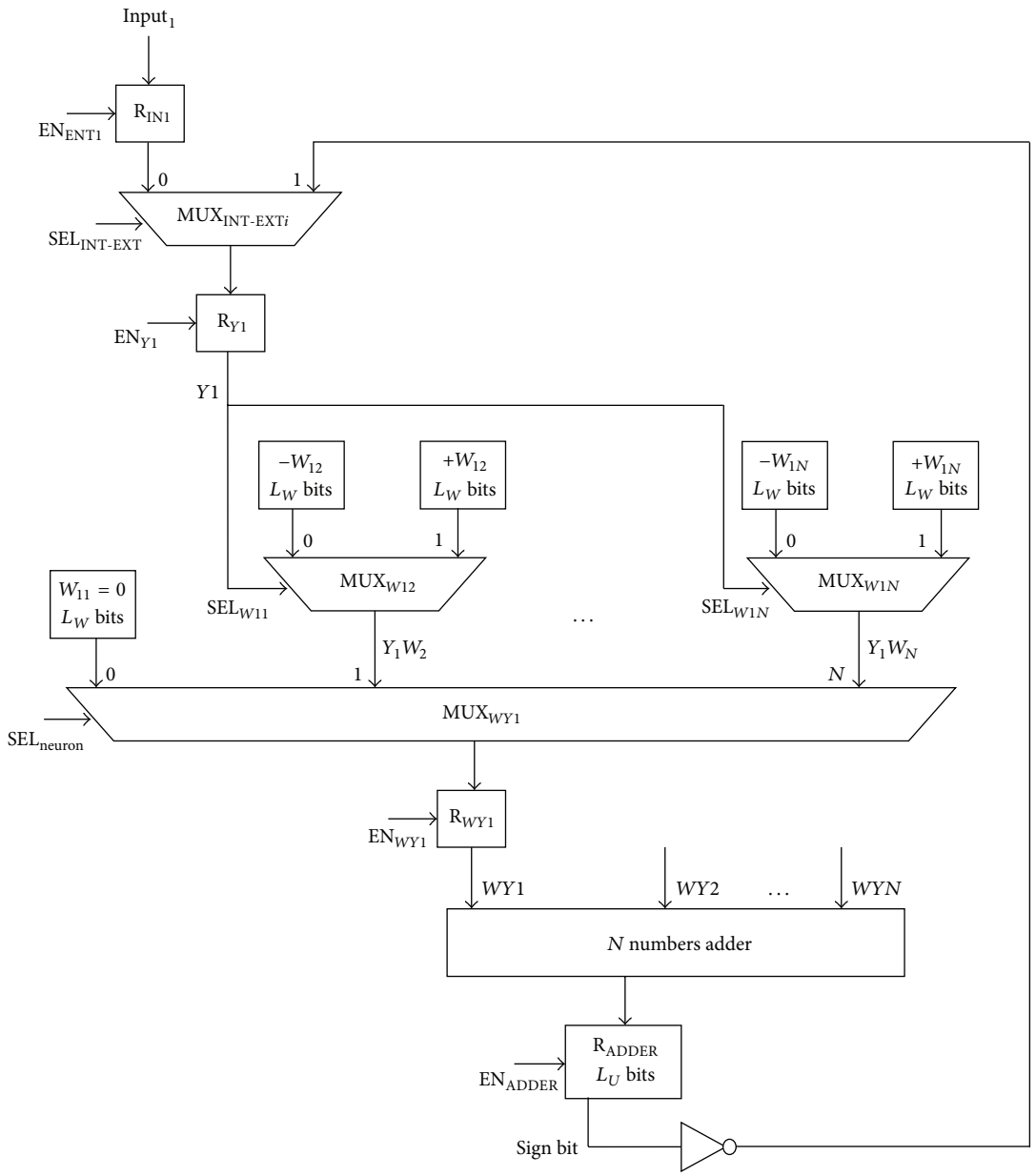

Figure 7: Proposed FPGA implementation of neuron #1.

weight value is $\pm M$, when all elements of the vector describing the stable states programmed contribute with identical values. Therefore, the length of weight registers (L_W) can be calculated by

$$L_W = 2 + \log_2 |M| . \tag{4}$$

The two bits added are due to the necessity of representing the actual number expressed by M and do not represent only M different states along with the weight signal. Equation (4) can be rewritten as a function of N, according to (3) by

$$L_W = 2 + \log_2 0.14N \tag{5}$$

with N always a positive number.

Also, in order to prevent overflows while registering the weighted summations, it is possible to calculate, from (2), the maximum value of U_i, in a network with N neurons, as $\pm M(N-1)$. From the capacity analysis described by (3), the number of bits to register U value can be written as a function of the quantity of neurons by

$$L_U = 1 + \log_2 \left(\left(N^2 - N \right) 0.14 \right), \tag{6}$$

where L_U denotes the length of U registers. The bit added is due to the representation of the U signal.

Also, in order to save space on the FPGA, only half of the w_{ij} are stored. Since, as mentioned earlier, $w_{ji} = w_{ij}$, the registered weights are used twice in the present implementation [13].

State 0: wait for **capture-button**
(i) *user input*
State 1: EN_{ENT} = **1**; $SEL_{INT\text{-}EXT}$ = **0**;
(i) *input data admission*
State 2: EN_Y = **1**; $SEL_{INT\text{-}EXT}$ = **0**;
(i) *set initial* V_i *state*
State 3: $EN_{BEGIN\text{-}EPOCH}$ = **1**; $SEL_{INT\text{-}EXT}$ = **1**;
(i) *register* V_i *state at the begin of an epoch*
(ii) $SEL_{INT\text{-}EXT}$ = *1 while* ***not*** *end*
State 4: $EN_{WY}i$ = **1**; $SEL_{INT\text{-}EXT}$ = **count_neuron**;
State 5: EN_{ADDER} = **1**;
State 6: $EN_Y i$ = **1**; if **count_neuron** = **neuron_size**
then **next state** = State 7;
else **next state** = State 4; ++ **count_neuron**; ++ *i*;
State 7: $EN_{END\text{-}EPOCH}$ = **1**;
(i) *register* V_i *state at the end of an epoch*
if **converge** = **1** then **next state** = State 0;
else **next state** = State 3;
end;

Algorithm 1: Pseudocode of control unit.

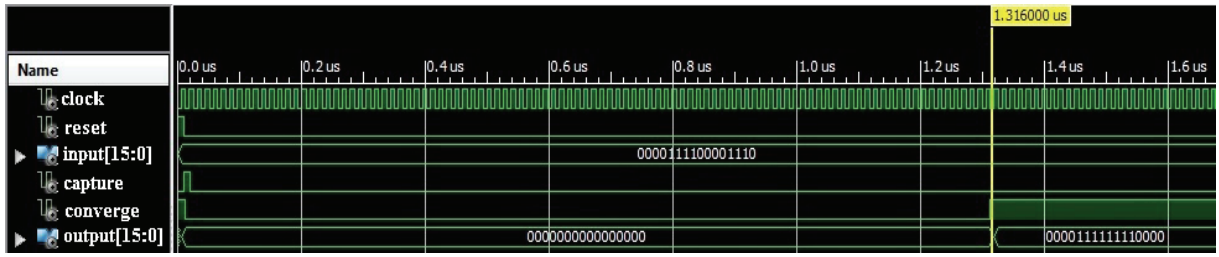

Figure 8: Simulation of the HNN, converging in approximately 1.3 μs to the stored stable state: "000011110000".

4. Results and Discussions

This section presents the set of experiments conducted in order to obtain some relevant parameters of the HNN implementation on FPGA, such as, maximum operating frequency and chip-area occupancy, in function of the quantity of neurons. The target device chosen to embed the network was the Spartan3E XC3S250E from Xilinx Inc. The choice was made aiming to employ a device with similar features to other published works [10, 13], that is, a FPGA with approximately equal numbers of logic cells and system gates.

The parameters set for the first experiment were $N = 16$ neurons, $M = 2$ patterns, $L_w = 3$ bits, and $L_u = 6$ bits. According to ISE Project Navigator 13.1 used to design the proposed architecture, the maximum frequency was 81.390 MHz and the digital system occupied 197 slices, which means 8% of the space available on the chip. Figure 8 contains the network response to a prompting input pattern with Hamming distance from a stable state equal to 7 corrupted bits in the prompting condition. The figure shows that the network reached the convergence in approximately 1.3 μs, which is equivalent to 100 clock cycles.

In order to allow a graphic visualization of other experiments conducted, Figures 9 and 10 are presented. In this experiment, the parameters were set to $N = 32$ neurons, $M = 4$ patterns, $L_w = 5$ bits, and $L_u = 9$ bits. Figures 9 and 10 depict two stable states stored by the HNN (left side of the pictures) and the prompting inputs applied to the network with some corrupted bits (right side of the pictures). The first stored pattern, in Figure 9, represents the letter "U" and the second stored pattern, in Figure 10, represents number "5." Both input patterns applied to the network have Hamming distance from stored patterns equal to 10 corrupted bits and were successfully restored by the developed system.

Table 1 shows the information obtained from the experiments conducted. The table contains the parameters N and M for the sequence of HNNs implemented and the information on maximum frequency and maximum output time after clock and number of occupied slices for each implementation. Also, in order to better visualize the data, Figures 11 and 12 illustrate the maximum frequency and the number of occupied slices graphically, according to the network size.

From the obtained data, it can be seen that the architecture developed takes, at most, 4.144 ns to produce an output after the clock input, independently of the network configuration. Such stability is due to the parallel architecture implemented, because increments of neurons to the network do not increase the depth of the proposed circuit and,

Table 1: Parameters obtained from the experiments.

Implementation	N—M (Quantity of neurons and stored patterns)	Maximum frequency (MHz)	Maximum output time after clock (ns)	Number of occupied slices
1	16—2	81.390	4.114	8%
2	17—2	62.530	4.114	17%
3	18—2	56.047	4.114	18%
4	19—2	56.182	4.114	19%
5	20—2	51.635	4.114	20%
6	21—2	51.750	4.114	21%
7	22—3	47.115	4.114	26%
8	23—3	47.124	4.114	32%
9	24—3	43.883	4.114	34%
10	25—3	42.754	4.114	39%
11	26—3	40.069	4.114	41%
12	27—3	40.138	4.114	45%
13	28—3	37.818	4.114	47%
14	29—4	39.200	4.114	18%
15	30—4	36.892	4.114	18%
16	31—4	35.462	4.114	20%
17	32—4	33.553	4.114	18%

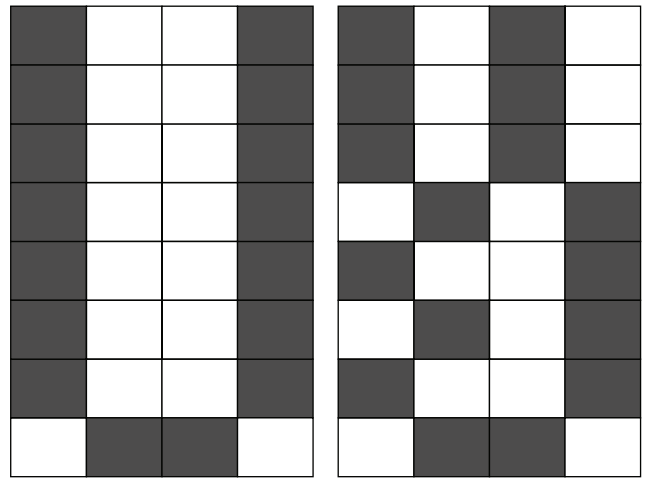

Figure 9: Illustration of one pattern memorized by the HNN representing letter "U" (left) and its corrupted version applied to the network (right). After convergence, the system outputted the stored pattern (left) successfully.

therefore, do not increase the maximum output time after clock.

As expected, a decrease of the system maximum input frequency according to the addition of neurons to the network can be seen in Figure 11. This decrement is due to a greater spread in the distribution of the FPGA resources caused by the increased number of logic elements and interconnections used in the network architecture.

The chip-area occupancy, however, presents a significant decrease between 13th and 14th implementation, despite the addition of a neuron to the network, as shown in Figure 12. Such occurrence is due to the increased number of patterns memorized by the network. The aggregation of one neuron, from 28 to 29 units, between experiments 13 and 14 (Table 1), allows the network to store one more pattern, rising from 3 to 4 stable states, which generates only even values of weights (no odd values are possible), according to (1). Even values represented in binary format have zeroes for the least significant bit. This enables the FPGA synthesis tool to reduce the entire logic chain employed to implement the presented architecture.

Lastly, an experiment was conducted in order to compare the proposed strategy of employing multiplexers instead of multipliers to calculate $W_{ji}V_i$ with common approach. The experiment consisted in implementing the target circuit of the weight unit (Section 3.1) using both architectures,

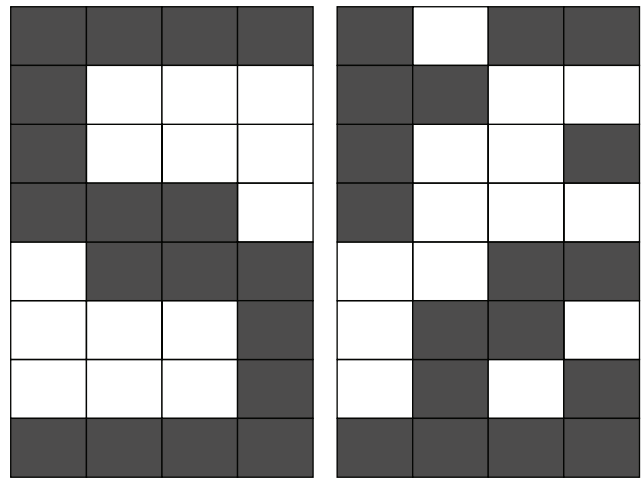

Figure 10: Illustration of one pattern memorized by the HNN representing number "5" (left) and its corrupted version applied to the network (right). After convergence, the system outputted the stored pattern (left) successfully.

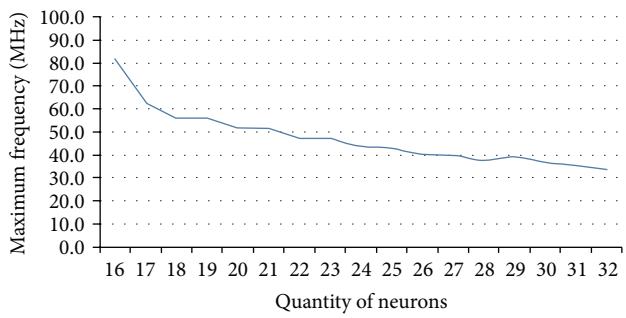

Figure 11: Maximum frequency of implementation.

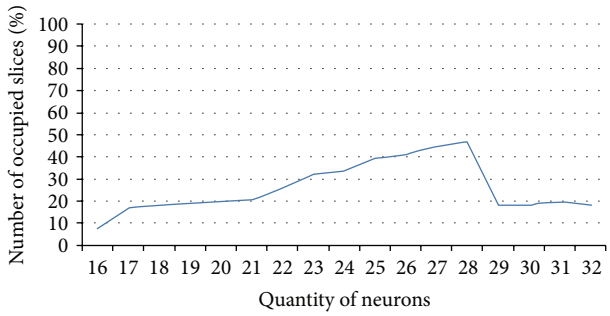

Figure 12: Chip-area occupancy.

multiplexers and multipliers, for the first and the last trials of Table 1. In the first implementation ($N = 16$ neurons, $L_w = 3$ bits), the use of multiplexers allows a reduction of 71 occupied slices on the chip. For the 17th implementation in Table 1 ($N = 32$ neurons, $L_w = 5$ bits), the proposed architecture allows a saving of 128 slices in the weight unit when compared to the use of common multipliers.

5. Conclusions

The present paper details a methodology for implementing the asynchronous version of the Hopfield Neural Network on an FPGA. The proposed architecture avoids the use of multipliers by using an array of multiplexers and does not store duplicated weights on the chip. Along with the proposal of the digital system, an approach to estimate the length of registers involved in the network design is presented. A set of experiments with the developed neural chip shows the decrease of the maximum operating frequency allowed in function of the quantity of neurons. On the other hand, an interesting observation is that the chip-area utilization does not always increase according to the enlargement of the network size; it also depends on other variables, as the weight matrix and the number of stored patterns of the HNN. The present purpose allows the implementation of larger networks on the chip by properly setting such parameters.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[2] S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, 3rd edition, 2009.

[3] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*, Springer, Dordrecht, The Netherlands, 2006.

[4] P. D. Moerland and E. Fiesler, "Neural network adaptations to hardware implementations," in *Handbook of Neural Computation*, IOP Publishing, Oxford University Publishing, New York, NY, USA, 1997.

[5] I. F. Saxena and E. Fiesler, "Adaptive multilayer optical neural network with optical thresholding," *Optical Engineering*, vol. 34, no. 8, pp. 2435–2440, 1995.

[6] H.-Y. Hsieh and K.-T. Tang, "Hardware friendly probabilistic spiking neural network with long-term and short-term plasticity," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 12, pp. 2063–2074, 2013.

[7] M. Papadonikolakis and C.-S. Bouganis, "Novel cascade FPGA accelerator for support vector machines classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 7, pp. 1040–1052, 2012.

[8] G. Borgese, C. Pace, P. Pantano, and E. Bilotta, "FPGA-based distributed computing microarchitecture for complex physical dynamics investigation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 9, pp. 1390–1399, 2013.

[9] R. N. A. Prado, J. D. Melo, J. A. N. Oliveira, and A. D. Dória Neto, "FPGA based implementation of a fuzzy neural network mo dular architecture for embedded systems," in *Proceedings of the IEEE World Congress on Computational Intelligence*, Brisbane, Australia, June 2012.

[10] B. J. Leiner, V. Q. Lorena, T. M. Cesar, and M. V. Lorenzo, "Hardware architecture for FPGA implementation of a neural network and its application in images processing," in *Proceedings of the 5th Meeting of the Electronics, Robotics and Automotive Mechanics Conference (CERMA '08)*, pp. 405–410, Morelos, Mexico, October 2008.

[11] M. Stepanova, F. Lin, and V. C.-L. Lin, "A hopfield neural classifier and its FPGA implementation for identification of symmetrically structured DNA motifs," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 48, no. 3, pp. 239–254, 2007.

[12] S. Saif, H. M. Abbas, S. M. Nassar, and A. A. Wahdan, "An FPGA implementation of a neural optimization of block truncation coding for image/video compression," *Microprocessors and Microsystems*, vol. 31, no. 8, pp. 477–486, 2007.

[13] W. Mansour, R. Ayoubi, H. Ziade, R. Velazco, and W. EL Falou, "An optimal implementation on FPGA of a Hopfield neural network," *Advances in Artificial Neural Systems*, vol. 2011, Article ID 189368, 9 pages, 2011.

[14] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 81, no. 10, pp. 3088–3092, 1984.

[15] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, 1982.

[16] D. J. Amit, H. Gutfreund, and H. Sompolinsky, "Storing infinite numbers of patterns in a spin-glass model of neural networks," *Physical Review Letters*, vol. 55, no. 14, pp. 1530–1533, 1985.

[17] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Santa Fe Institute Studies in the Sciences of Complexity. Lecture Notes, I, Addison-Wesley, Redwood City, Calif, USA, 1991.