A new Spiking Neural Network with Extreme Learning for FPGA implementation

1st Iván R. Peralta
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

2nd Nanci Odetti
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

3rd Eduardo Filomena
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

4th Juan I. Rufiner
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

5th Nahuel Ricart
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

6th Hugo L. Rufiner
Instituto Sinc(i) & Laboratorio de Cibernética
FICH-UNL-CONICET & FI-UNER
Santa Fe, Argentina
lrufiner@sinc.unl.edu.ar

***Abstract*—This paper proposes a parallel fixed point spiking neural network (SNN) implemented in a field programmable gate array (FPGA) with spike response neuron model that makes use of concepts related with extreme learning machines (ELM). For this reason the network is called "Digital Extreme Learning Spiking Neural Network" (DELSNN). The network was designed specifically for its implementation in FPGA. Internal structure and mode of operation are described, where the capability of processing continuous spikes is demonstrated. Matlab numerical calculation software is used for computational model development and training. Then a VHDL model for final implementation in FPGA is generated. For the sake of brevity in this paper only FPGA implementation aspects of DELSNN are presented and discussed. The entire VHDL project was developed using the Xilinx ISE platform, and an Atlys kit board which has a Spartan-6 LX45 as the target FPGA.**

***Index Terms*—Spiking Neural Networks, Extreme Learning Machines, FPGA, VHDL.**

I. Introduction

Artificial neural networks are inspired by their biological counterparts and are composed of basic units called neurons. They are interconnected by different weights which represent the intensity of the interaction between each other. Currently, deep artificial neural networks achieve the best performance in virtually any machine learning benchmark ranging from speech recognition to natural language processing to name a few applications [2]. The enormous number of operations that deep artificial neural networks require, represents a great drawback. This limits its execution on platforms with limited processing and energy resources. Currently, their execution is offloaded to remote computer clusters or GPUs. Nevertheless, many applications require fast responses and low-energy consumption as crucial features, for example speech processing in mobile platforms or robotic systems.

Spiking Neural Networks (SNNs) appear as an interesting alternative to simulate large-scale neural networks in real time applications [4], [5]. Actually, neuromorphic engineering [1] allows the emulation of SNNs on hardware in real-time with much higher efficiency in terms of power and speed compared to conventional computing platforms. An example is TrueNorth platform, which is capable of stimulating a million spiking neurons in real-time with only 3 mW of power consumption. The counterpart network on a traditional computing platform was 100, 200 times slower and consumed 100.000 to 300.000 times more energy in each synaptic event [15].

The SNNs were create two decades ago, as the result of searching for a closer analogy to biological neurons. The use of SNN is of special interest in applications that involve a temporal dynamic in the problem to be solved. These have been used on recent applications in many areas of pattern recognition such as visual processing [6], [7], speech recognition [8], [9], [16], and medical diagnosis [10], [11], among others [12], [13]. The aforementioned networks faithfully reproduce the neural biological systems with two effects. On the one hand they try to imitate the transfer of information between neurons through pulses called spikes, in the way it happens in the biological synapses with Action Potentials. Since the neurons communication is done through spikes ('1') or non-spikes ('0'), they are well suited to be implemented on digital hardware. On the other hand, dynamic processing of the signals inside the neurons is done [14] and multiple delayed synaptic terminals can be applied between neurons connections [17]. Thus, they are good candidates to temporal patterns classification.

Aside from the large neuromorphic structures named above, many custom implementations of SNN have been presented in analog and digital hardware [3]. Different architectures have been shown, ranging from wide framework [18], [19] capable to process great custom networks ($> 10,000$ neurons) at many times the real-time speed, to compact systems where small networks are directly implemented onto hardware [20]–[23]. This latter one is used to apply custom solutions to small problems or conform preliminary studies that allow construction of more great SNNs on future.

In an SNN, weighted spikes at the inputs of a neuron are integrated over time and when that integral exceeds a certain

threshold, a corresponding spike is generated at the neurons output. During the training procedure, the strength of the synaptic weights can be changed as result of learning rule application. Actually, a great drawback for use SNNs is their training. For a recent review in learning rules in SNNs, see [5]. One recent paradigm that comes from machine learning community is the Extreme Learning Machine (ELM) [24]. In this structure the connections between the input and hidden layers are randomly weighted, and fixed (they are not altered during training). This strategy is very useful in classification tasks because only the weights of a single layer are trained. The purpose of random weights is to project the signals coming from the first layer in a larger space (by a complex non-linear transformation). The classes that are difficult to separate in the original space are projected to a larger space to allow them to be easily separable in the new representation space.

In this paper we present DELSNN (Digital Extreme Learning Spiking Neural Network) and its compact FPGA implementation. It was designed as an alternative SNN for patterns classification in a supervised manner, using concepts related to ELM. This paper describes the internal structure and mode of operation of the aforementioned network, together with its implementation in FPGA. Due to the require length constraints of this paper, the training algorithm and its applications will not be shown. The reader is referred to [29] to delve into these aspects of the network.

II. DELSNN

The proposed SNN has a *feedforward* architecture with two layers of neurons (**I** and **J**) and an input pattern that can be reduced, for a given instant, to a vector **V** of N components (see Fig. 1). Each neuron of **I** layer connects to each vector elements V_v and each neuron in output layer **J** receives connections from all neurons of layer **I**. Connections between layers are composed of several delay propagation lines weighted by a particular weight for each delay. In one connection between first layer (**I**) and the second layer (**J**), the k-th connection delay between i and j neurons is characterized for a d^k delay and a weight W_{ij}^k (see Fig. 1). The network is designed to perform data classification (encoded in spikes) which are grouped into different classes. In a trained DELSNN, each output neuron represents a possible class and if an example of that class is presented to the network, only the neuron associated with that class must emit spikes. The weights that link the first and second layer of neurons are set randomly, while the weights of the second layer are trained with the examples of the training set. It is worth noting that DELSNN was designed from the beginning with its FPGA implementation in mind, hence explanation of its mode operation is will done in the domain of discrete time.

A. Neuronal Model

Diverse neuron models have been proposed such as the spike response model (SRM) [25], the Izhikevich neuron model [26], and the leaky integrated-and-fire (LIF) neuron

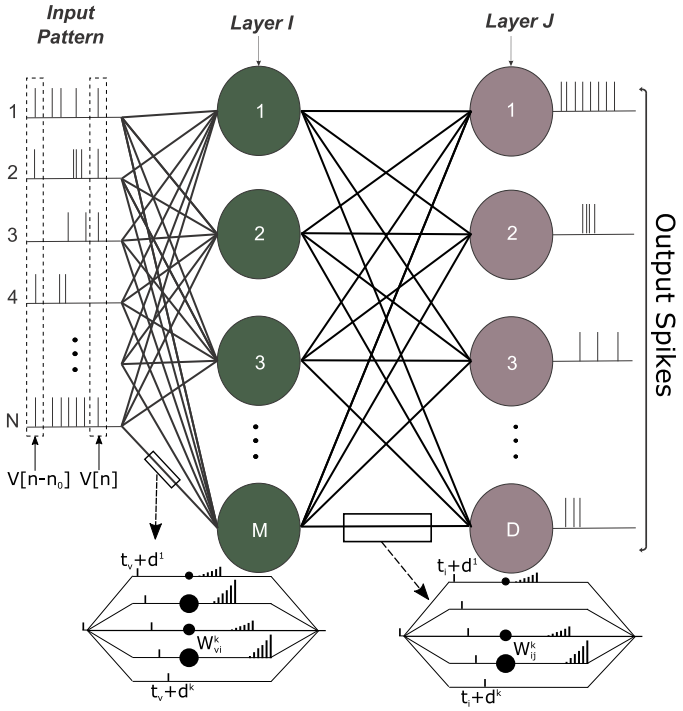

Fig. 1. Internal structure of DELSNN.

[27]. The neuronal model used for this work is a variant of the impulse response model (SRM). Below, the operation of a neuron i belonging to the input layer is described, but this explanation can be extended to any neuron of the SNN. The state of the neuron i is described by the state variable u_i. This neuron fires if u_i reaches the threshold ϑ. At the instant that threshold is crossed, the discrete firing time $n_i^{(f)}$ is defined, where f indicates the firing number or spike emitted by the neuron i. The set of all firing times of the neuron i is defined by:

$$\mathcal{F}_i = \left\{ n_i^{(f)} \right\} = \{ n / u_i[n] > \vartheta \} . \tag{1}$$

A sequence of spikes emitted by a neuron of the layer **I** can be written as a sequence of discrete Dirac impulses:

$$S_i[n] = \sum_{n_i^{(f)} \in \mathcal{F}_i} \delta[n - n_i^{(f)}], \; with \; \mathcal{F}_i = \left\{ n_i^{(1)}, ..., n_i^{(s)} \right\} . \tag{2}$$

The inputs to the neuron i are given by the spikes coming from each component of the input vector **V**. In turn, since the input vector is updated at each simulation instant, the time evolution of each **V** component can be described as $V_v[n]$, where n is the simulation time and v (with $1 \leq v \leq N$) indicates the v-th vector component.

A spike that belongs to an element V_v of the input vector is distributed in each of the propagation delays d^k of each connection that leaves that element. Once the spikes have been delayed by the propagation delays, they arrive at the neurons as presynaptic pulses that determine a change in the neuronal state. Each presynaptic spike that enters the neuron i, increases (or decrements) its variable u_i in an amount $W_{vi}^k \, h[n - d^k]$,

where h is the function of impulse response (spike) of the neuron. The increase or decrease of the variable depends on whether the weight is positive or negative, respectively. The state u_i of neuron i at time n is given by the linear superposition of contributions of all propagation delays of all components of input vector: [1]

$$u_i[n] = \sum_{\nu=1}^{N} \sum_{k=1}^{K} \sum_{\tau=0}^{\infty} V_\nu[\tau]\, W_{\nu i}^k\, h[n-\tau-d^k]. \qquad (3)$$

If we consider a neuron of the output layer **J**, the update equation of the state variable u_j of neuron j at time n is given by the linear superposition of contributions of all spikes coming from neurons of layer **I**:

$$u_j[n] = \sum_{i=1}^{M} \sum_{k=1}^{K} \sum_{\tau=0}^{\infty} S_i[n]\, W_{ij}^k\, h[n-\tau-d^k]. \qquad (4)$$

Like the Ec. (2), a sequence of spikes emitted by a neuron of the **J** layer can be written as a sequence of discrete Dirac impulses:

$$S_j[n] = \sum_{n_j^{(f)} \in \mathcal{F}_j} \delta[n - n_j^{(f)}], \qquad \mathcal{F}_j = \left\{ n_j^{(1)}, ..., n_j^{(s)} \right\}, \quad (5)$$

where $n_j^{(f)}$ is the instant of firing number f of neuron $j \in \mathbf{J}$.

Taking into account Ec. (3) and (4), one can interpret the internal functioning of a neuron, that is, the variation of its state u, as the discrete linear convolution of incoming spike sequences with an impulse response function $h[n]$. This implies that at the moment of simulating the SNN, we can choose a $h[n]$ with arbitrary form, which is a potentiality of SRM model.

III. ARCHITECTURE AND IMPLEMENTATION

A complete system diagram is shown in Fig. 2. The SNN is implemented within the FPGA, whose operation is controlled by the control unit (GENERAL CONTROL UNIT). This unit is in charge of controlling and exchanging the incoming and outgoing SNN spikes with the PC through a UART block (Universal Asynchronous Receiver-Transmitter).

For each simulation step, the control unit places in its output *vector_in* a binary vector with the input spikes to the SNN. Then activates the signal *clk_spk*. The SNN processes the input and places in its output *spikes_out* another binary vector with the responses of each of the output neurons for that simulation step. Then, the SNN activates its output *end_processing_SNN* to indicate to control unit that it has finished processing the entries. The control unit is responsible, through the UART, to send these outputs to the PC and to receive the next inputs for the next simulation step.

A *pushbutton* is used to generate a reset signal for the entire system. The *leds* of the board are also used to report possible problems or errors in data transmission.

[1]The internal sum with the upper end ∞ indicates that the sum is done until the end of the whole input pattern.

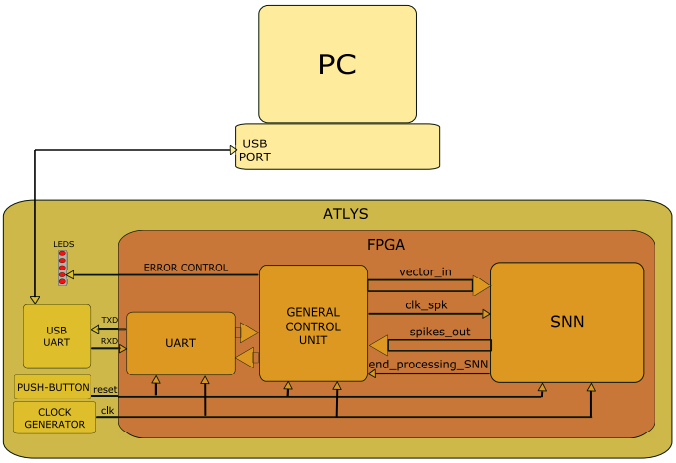

Fig. 2. Block diagram of the complete system implemented in the FPGA, *Atlys* kit board and communication with the PC.

The main objective of this work is to achieve a functioning of the SNN in the FPGA but without emphasizing in the efficiency of operation or the optimal use of FPGA resources. So that some simplifications are made in order to reduce the design complexity: a) The main simplification is that one processing element (PE) is used for each neuron (see [29]). This implies that each PE will only control a single neuron, which significantly reduces the number of neurons that can be implemented in the FPGA. b) The internal registers of each neuron are represented by means of 32-bit integers, which allows the direct use of the data type *integer* of the VHDL language. This simplifies the neuron design, but does not make optimal use of the device resources.

A. Neural Design

The SNN implementation is carried out following a Serial Processing Parallel Arithmetic (SPPA) architecture. That is, an arithmetic is used at the level of bit registers and a serial processing of each connection that enters a neuron. Given that we are in an experimental stage in which we wish to evaluate the SNN performance in response to different responses to the impulse $h[n]$, the architecture designed must be capable of representing any model of neuronal response (SRM).

The SRM model is not a simple model to implement, so there are not many works that have addressed this type of neural model in compact design implementations [20], [23], [28]. The main drawback of this model is that to calculate the membrane potential, the impulse response function must be multiplied by the interconnection weight of each synapse that has an active spike. In addition, in order to simulate the temporal evolution of each connection, the postsynaptic potentials (PSP) of each synapse must be treated in an independent registry, which in a SNN with many connections per neuron, would demand many device resources. Another important aspect is to determine if the neural model supports the application of a spike in its input while the evolution of a previous PSP is occurring. In an extreme case the

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
I. R. Peralta, N. Odetti, E. Filomena, J. Rufiner, N. Ricart & H. L. Rufiner; "A new Spiking Neural Network with Extreme Learning for FPGA implementation"
Proceedings of the X Southern Conference on Programmable Logic - SPL 2019, apr, 2019.

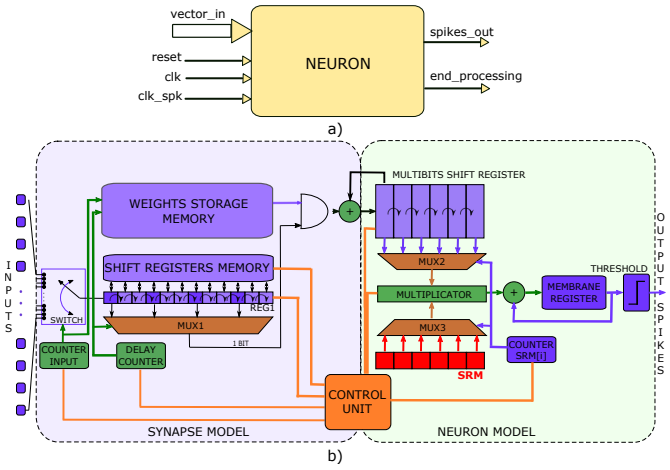

Fig. 3. Neuronal model implemented in the FPGA. (a) Incoming and outgoing signals from a neuron. (b) Simplified schematic diagram of the synapse and neuron model. Thick lines represent multi-bit connections, while thin lines represent one-bit connections.

neuron must support the application of multiple input spikes applied consecutively over time. This is important, since if this capacity is not available, the type of possible coding for the entry patterns to the SNN is limited.

The design proposed in this work notably reduces the complexity of implementation through the use of accumulators and shift registers to represent the evolution of the membrane potential. In addition, it has the ability to apply consecutive spikes over time. This capacity has not been evidenced in any of the models of SRM neurons implemented in FPGA described so far. The Fig. 3 describes the input and output signals of the neuronal model and its internal functioning.

Each neuron receives: the vector *vector_in*, with the input spikes corresponding to a simulation step; the signal *clk_spk*, which defines the simulation period; the signal *clk*, that determines the internal working speed of the neuron and has a much shorter period than *clk_spk* (typical ratio of 1: 10000), and the *reset* signal to manually restart the system. On the other hand, the neuron has two single-bit outputs. The output *spikes_out* describes the outgoing spikes of the neuron and connects itself, depending on the neuronal layer to which it belongs, to the inputs of its postsynaptic neurons or to the vector *spikes_out* that comes out of the SNN described in Fig. 2. It also has as output the signal *end_processing*, which indicates that the neuron has finished processing its inputs. [2] Once all neurons of SNN have finished processing, the signal *end_processing_SNN*, described above, is activated, indicating the end of the processing of the complete network for a particular input vector.

Pressing the *reset* button starts the system. Initially the control unit puts all registers, counters and values of the SHIFT REGISTERS MEMORY to zero, to allow a simulation with null initial conditions. Then, the control unit waits for the activation of the input signal *clk_spk*, which is issued by the GENERAL CONTROL UNIT. This signal tells the SNN to begin processing each of the connections with the presynaptic neurons (in the case that it is a neuron of the last layer) or the input vector (for the neurons of the first layer) as appropriate. Then the first bit of the shift register REG1 is loaded with the logical value present in the first input. The selection of the entry to be processed is done through the COUNTER INPUT. The REG1 determines the propagation delay of a particular input of the *vector_in*. Once the REG1 register has been modified, it is stored in the SHIFT REGISTERS MEMORY for its later reuse in the processing of the next pattern. To map the weights, some outputs of REG1, with specific delays, are connected to a multiplexer MUX1. Then, the output of MUX1 is combined by an AND function with the weight corresponding to that delay. In such a way that if there is a '1' in the selected delay, the weight for that delay will accumulate in the first sub-register of the MULTIBITS SHIFT REGISTER. The weights of the connections are stored in the WEIGHTS STORAGE MEMORY and the address of that memory is determined by the DELAY COUNTER and the COUNTER INPUT. The process described above is repeated until all the inputs of the neuron are scanned.

Once all the inputs have been processed, the membrane potential stored in the MEMBRANE REGISTER is calculated. For this moment, in the first sub-register of the MULTIBITS SHIFT REGISTER, a weight value is stored, which is the sum of all the weights of the active delays associated with each input of the neuron. The impulse response $h[n]$ is stored in the SRM register. Therefore, the first value of the impulse response (SRM[1]) multiplied by said total weight is added to the membrane potential. Then there is a shift to the right of the values of each sub-register of the MULTIBITS SHIFT REGISTER. This allows that in the processing of the next input vector to the neuron, the initial accumulation of weights is not lost and the complete evolution of the PSP associated with that first input vector can be described. Therefore, multiplication is done for all sub-registers of MULTIBITS SHIFT REGISTER. That is, in each processing of an input vector, the multiplication of the i-th sub-register of the MULTIBITS SHIFT REGISTER with the i-th value of the impulse response (SRM [i]) is added to the membrane potential. The MULTIBITS SHIFT REGISTER is what allows the neuron to process contiguous spikes at the entrance of the neuron. Finally, if the MEMBRANE REGISTER exceeds a pre-set threshold, the *spikes_out* output of the neuron is activated.

The CONTROL UNIT, internal to the neuron, was carried out by means of a state machine implemented through a process in VHDL. Its operation is timed by the fast clock signal *clk* and is responsible for both the control of the synaptic model and the neuronal model.

Both the MULTIBITS SHIFT REGISTER and the SRM are composed of 32-bit sub-registers (integers). The SRM register is identical for all neurons and is defined in a configuration file (*configuration.vhd*) together with other design parameters.

The innovation of the design proposed in this paper is that

[2]This output must be differentiated from the output *end_processing_SNN*, which indicates when the complete SNN has finished processing the entries.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
I. R. Peralta, N. Odetti, E. Filomena, J. Rufiner, N. Ricart & H. L. Rufiner; "A new Spiking Neural Network with Extreme Learning for FPGA implementation"
Proceedings of the X Southern Conference on Programmable Logic - SPL 2019, apr, 2019.

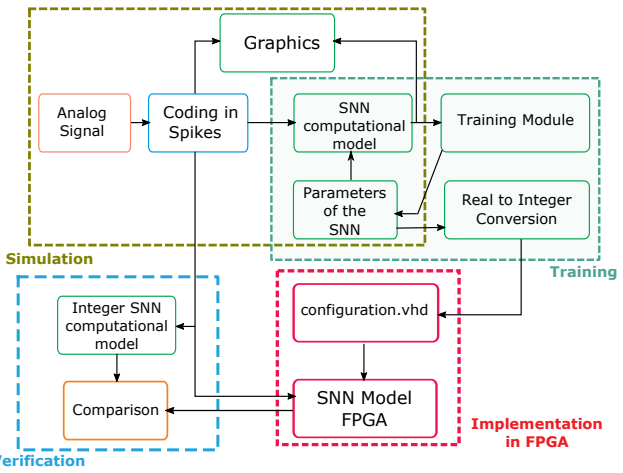

Fig. 4. Conceptual diagram of the different stages of work.

it does not need independent registers to store, separately, the PSP of each entry as it happens in [23]. Instead, an integral treatment of the inputs is carried out, being able to keep all information of the membrane potential in the MULTIBITS SHIFT REGISTER and the MEMBRANE REGISTER. As evidenced, the proposed model uses a serial processing of the neuron inputs, which reduces the complexity of the hardware significantly. However, all neurons are processed in parallel, which allows obtaining a good relationship between the performance and the occupation of the device.

B. SNN module

A block of higher hierarchy than the neurons (called SNN) is defined, which groups all of them into a single structure. To generate said block in VHDL, the neurons are replicated by means of a loop (*for ... generate*) within which the connections between each neuron are defined. The replication of each neuron is done through the use of generic parameters, which are stored in the file *configuration.vhd*. In turn, in this block the signal *end_processing_SNN* is generated by applying an AND logical function of all signals *end_processing* belonging to network neurons.

The storage of the weights of each connection is done in a BROM block, one for each neuron.

IV. Performance Evaluation

The Fig. 4 shows a conceptual scheme of the different stages to evaluate the correct functioning of the SNN. Firstly, a computational model of the SNN was implemented and its operation was evaluated with the excitation of different spike patterns. The computational model was made in MATLAB and uses a floating type arithmetic (real numbers). This model describes the operation of the network, emulating the operation of the SNN implemented in VHDL. Then the weight values are determined. The training of the network is done iteratively in MATLAB (see [29]).

Once the training stage is finished, given that the SNN model in VHDL was programmed with integer arithmetic, it is necessary to convert the values of the weights from real numbers to integers. In Ec. (6) the conversion of the real type weights to the integer type is described.

$$weight_{int} = round\left(\frac{(N_c - 1) * weight_{float}}{MAX - MIN}\right), \quad (6)$$

where N_c is the amount of levels used in the quantization. The values MAX and MIN are the respective maximum and minimum for the whole set weights of the SNN once the training is finished. The $round(x)$ function rounds the value of x to the nearest integer.

The Ec. (6) not only modifies the decimal part of the weights, but also modifies the scale of them, so it is necessary to apply the same transformation to the thresholds of fire.

The representation of signed numbers that is used in this work is complement to 2 (one bit to represent the sign and B bits for the magnitude). The number of magnitude bits and the number of levels (N_c) are related by the Ec. (7),

$$B = \log_2(N_c). \quad (7)$$

Once the conversion of the weights is done, we proceed to create the file *configuration.vhd*, which is a text file that groups all configuration parameters of the SNN. The .vhd extension allows it to be integrated into the Xilinx Project Navigator project without any previous steps. In this way, the file *configuration.vhd* becomes the nexus between the computational design in MATLAB of the SNN and the design in the Project Navigator of Xilinx.

Once the design of the SNN is implemented in the FPGA, it is necessary to verify its correct operation. To carry out this verification, a MATLAB *script* was implemented that stimulates with a single pattern the two models of SNN: the computational quantized and the electronic implanted in the FPGA. After exciting both networks, the *script* picks up the output spike sequences of each model and compares them. It is determined that the operation is correct if both responses are equal. The spike responses to same stimulus of both the computational model with its quantized weights (MATLAB) and the electronic model (FPGA) were identical in all experiments, corroborating the correct functioning of DELSNN.

V. Results

Although this paper is limited to describing the implementation of DELSNN in FPGA, it is worth mentioning that in [29] the performance of DELSNN in the task of speech recognition for the classification of isolated digits is shown with results comparable to other state of the art alternatives ($WER(\%) = 12,75$). In most experiments only a 5 bits weights quantization was necessary to not affect significatively the network recognition.

The design in VHDL is completely synthesizable on the FPGA, as an example, the Spartan 6 XC6SLX45 device is used with 8-bits weights representation. For comparative purposes Table I shows the resources used in different DELSNN structures. Due to 8-bit quantization, most of the BRAMs were

TABLE I
COMPARISON OF LOGIC RESOURCES UTILIZATION IN DIFFERENT NETWORKS STRUCTURES WITH 8-BITS WEIGHTS REPRESENTATION FOR K=3 AND K=10 DELAYS / WEIGHTS PER CONNECTION.

DELSNN Structure	Synapses	Slice Registers	Slice LUTs	Block RAM
2x2x2	24	1599	3152	4
	80	2271	3641	6
9x9x4	351	5112	10028	13
	1170	7296	11673	20
32x4x16	576	7145	11312	20
	1920	10569	14188	30

used by the SHIFT REGISTERS MEMORY, which are used for the models of synapses between neurons. For the 32x4x16 and 1920 synapses case, only 30 of the 116 available BRAMs were used. The large number of slices used is mainly due to the fact that all the internal registers of the design are of an integer type (32 bits). It should be noted that this excessive bit resolution is not necessary for most of the internal registers of neurons. All the cases in the table I were tested with a clock frequency of 100 MHz and the worst case limits the maximum possible frequency to 117MHz.

VI. CONCLUSIONS

This work presents a proposal for the parallel fixed point implementation of a new type of SNN in a FPGA. The internal architecture of SNN implementation was shown and analysed. FPGA implementation was carried out satisfactorily, remarking an innovative neural design able to perform continuous spikes processing and low amount of bits required for weigths representation on internal registers. The results obtained were highly satisfactory, indicating the potential feasibility of the technique for use in practical situations. Since that the optimization of the FPGA resources was not a direct goal of this work, it is also possible to perform an explicit optimization of the VHDL design to improve its neurons storage capacity and processing speed in future works.

REFERENCES

[1] Indiveri, G., Horiuchi, T. K. (2011). Frontiers in neuromorphic engineering. Frontiers in neuroscience, 5, 118.

[2] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

[3] Misra, J., Saha, I. (2010). Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing, 74(1-3), 239-255.

[4] Ponulak, F., Kasinski, A. (2011). Introduction to spiking neural networks: Information processing, learning and applications. Acta neurobiologiae experimentalis, 71(4), 409-433.

[5] Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., Maida, A. S. (2018). Deep Learning in Spiking Neural Networks. arXiv preprint arXiv:1804.08150.

[6] Wysoski, S. G., Benuskova, L., Kasabov, N. (2010). Evolving spiking neural networks for audiovisual information processing. Neural Networks, 23(7), 819-835.

[7] Meftah, B., Lezoray, O., Benyettou, A. (2010). Segmentation and edge detection based on spiking neural network model. Neural Processing Letters, 32(2), 131-146.

[8] Tavanaei, A., Maida, A. (2017, November). Bio-inspired multi-layer spiking neural network extracts discriminative features from speech signals. In International Conference on Neural Information Processing (pp. 899-908). Springer, Cham.

[9] Wade, J. J., McDaid, L. J., Santos, J. A., Sayers, H. M. (2010). SWAT: a spiking neural network training algorithm for classification problems. IEEE Transactions on Neural Networks, 21(11), 1817-1830.

[10] Kasabov, N., Feigin, V., Hou, Z. G., Chen, Y., Liang, L., Krishnamurthi, R., ... Parmar, P. (2014). Evolving spiking neural networks for personalised modelling, classification and prediction of spatio-temporal patterns with a case study on stroke. Neurocomputing, 134, 269-279.

[11] Ghosh-Dastidar, S., Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. Neural networks, 22(10), 1419-1431.

[12] Borisyuk, R., Chik, D., Kazanovich, Y., da Silva Gomes, J. (2013). Spiking neural network model for memorizing sequences with forward and backward recall. BioSystems, 112(3), 214-223.

[13] Kasabov, N., Dhoble, K., Nuntalid, N., Indiveri, G. (2013). Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. Neural Networks, 41, 188-201.

[14] Gerstner, W., Kistler, W. M. (2002). Spiking neuron models: Single neurons, populations, plasticity. Cambridge university press.

[15] Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., ... Brezzo, B. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. Science, 345(6197), 668-673.

[16] Mir-Amarante, L., Gmez-Rodrguez, F., Jimnez-Fernandez, A., Jimnez-Moreno, G. (2017). A spiking neural network for real-time Spanish vowel phonemes recognition. Neurocomputing, 226, 249-261.

[17] Natschlger, T., Ruf, B. (1998). Spatial and temporal pattern analysis via spiking neurons. Network: Computation in Neural Systems, 9(3), 319-332.

[18] Schoenauer, T., Mehrtash, N., Jahnke, A., Klar, H. (1999, March). MASPINN: Novel concepts for a neuroaccelerator for spiking neural networks. In Ninth Workshop on Virtual Intelligence/Dynamic Neural Networks (Vol. 3728, pp. 87-97). International Society for Optics and Photonics.

[19] Hellmich, H. H., Geike, M., Griep, P., Mahr, P., Rafanelli, M., Klar, H. (2005, July). Emulation engine for spiking neurons and adaptive synaptic weights. In Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on (Vol. 5, pp. 3261-3266). IEEE.

[20] Iakymchuk, T., Rosado, A., Frances, J. V., Batallre, M. (2012, July). Fast spiking neural network architecture for low-cost FPGA devices. In Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on (pp. 1-6). IEEE.

[21] Moctezuma, J. C., McGeehan, J. P., Nunez-Yanez, J. L. (2015). Biologically compatible neural networks with reconfigurable hardware. Microprocessors and Microsystems, 39(8), 693-703.

[22] Wan, L., Luo, Y., Song, S., Harkin, J., Liu, J. (2016, June). Efficient neuron architecture for FPGA-based spiking neural networks. In Signals and Systems Conference (ISSC), 2016 27th Irish (pp. 1-6). IEEE.

[23] Rosado-Muoz, A., Fijakowski, A. B., Bataller-Mompen, M., Guerrero-Martnez, J. (2011, January). FPGA implementation of spiking neural networks supported by a software design environment. In Proceedings of 18th IFAC World Congress. Milano, Italy Milano, Italy.

[24] Huang, G. B., Zhu, Q. Y., Siew, C. K. (2006). Extreme learning machine: theory and applications. Neurocomputing, 70(1-3), 489-501.

[25] Jolivet, R., Timothy, J., Gerstner, W. (2003). The spike response model: a framework to predict neuronal spike trains. In Artificial Neural Networks and Neural Information ProcessingICANN/ICONIP 2003 (pp. 846-853). Springer, Berlin, Heidelberg.

[26] Izhikevich, E. M. (2003). Simple model of spiking neurons. IEEE Transactions on neural networks, 14(6), 1569-1572.

[27] Delorme, A., Gautrais, J., Van Rullen, R., Thorpe, S. (1999). SpikeNET: A simulator for modeling large networks of integrate and fire neurons. Neurocomputing, 26, 989-996.

[28] Johnston, S., Prasad, G., Maguire, L., McGinnity, M. (2005, September). Comparative investigation into classical and spiking neuron implementations on FPGAs. In International Conference on Artificial Neural Networks (pp. 269-274). Springer, Berlin, Heidelberg.

[29] Peralta Iván R. (2017). Reconocimiento de dígitos mediante Redes Neuronales Pulsantes implementadas en FPGA (Tesis de maestría). Facultad de Ingeniería, Universidad Nacional de Entre Ríos (FIUNER), Paraná, Argentina.

sinc(*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
I. R. Peralta, N. Odetti, E. Filomena, J. Rufiner, N. Ricart & H. L. Rufiner; "A new Spiking Neural Network with Extreme Learning for FPGA implementation"
Proceedings of the X Southern Conference on Programmable Logic - SPL 2019, apr, 2019.