SoC Kohonen Maps Based on Stochastic Computing

Alejandro Morán
Department of Physics
University of the Balearic Islands
Palma of Majorca, Spain
a.moran@uib.eu

Josep L. Rosselló
Department of Physics
University of the Balearic Islands
Palma of Majorca, Spain
j.rossello@uib.es

Miquel Roca
Department of Physics
University of the Balearic Islands
Palma of Majorca, Spain
miquel.roca@uib.es

Vincent Canals
Department of Physics
University of the Balearic Islands
Palma of Majorca, Spain
v.canals@uib.es

***Abstract*—Mobile systems and by extension Internet of Things (IoT) applications requests more and more Machine Learning functions, thus requiring a big computational power with a small power available. These demands have led to renewed interest in unconventional hardware computing methods capable of to implement complex functions in a simple way, with a very small power consumption. This work proposes a novel System-on-Chip (SoC) implementation of a Kohonen Map based on stochastic computing. In turn, to support this development, several stochastic block designs are presented as Winner-Take-All (WTA) similarity check and the squared Euclidian distance. The capabilities and performance of the proposed SoC solution is tested over a well-known classification task over the Fisher's Iris data set, archiving the same classification performance than the software. The proposed solution requests few hardware resources and low power, due to its inherent capacity to implement complex functions in a simple way. This enables to implement large self-learning classifiers based on Kohonen maps on tiny systems.**

Keywords—Stochastic logic, Unsupervised learning, Kohonen maps, Field programmable gate arrays, System-on-chip

I. Introduction

Internet of Things (IoT) devices span a wide range of application domains including manufacturing, personal wearables, logistics, smart-grid and agriculture applications [1]. At the same time, IoT requires diverse technology and specialized skill areas such as specialized hardware and sensor development, along with sophisticated real-time embedded firmware, cloud applications, Big Data analytics and Machine Learning (ML) [2] for massive real-time data into usable information, delivery of data to human-scale and human-usable platforms. The gradual integration of IoT devices with cloud computing moves intelligence [3] to edge devices and allows global decision-making based on local measurements. These new capabilities demand new high performance and low-power platforms to meet computational needs of data-intensive applications. Generally, these high performance platforms consist of a System-on-Chip (SoC) that incorporates embedded GPUs, DSPs and FPGA [4] in order to balance power, performance, size and cost. Recent advances in Hardware Description Language (HDL) synthesis [5] have improved the FPGA programming, which simplifies the use of FPGA's in SoC based applications.

Moreover, ML is vast and its applications are expanding rapidly with the emergence of the IoT devices that also have access to cloud computing. ML algorithms [6], [7] can be broadly classified into three categories based on the properties, mechanism of learning and the way data are used: supervised, semi-supervised and unsupervised algorithms. These techniques demonstrate unprecedented performance in solving complex real world problems in which the traditional approaches are not feasible or effective [8]. To support this technological revolution, hardware companies are racing to build CPU, GPU, SoC, tools, and frameworks that enables to achieve high computing capabilities with low power consumptions. In turn, silicon companies are focusing on alternative unconventional computation methods to circumvent the technological limits of the actual semiconductor industry [9] and obtain greater computational capabilities per watt. This is the case of probabilistic or stochastic computing architectures [10]–[13], which apply probabilistic laws to digital logic gates, thus performing pseudo analog operations with stochastic digital signals.

This work proposes a SoC solution to accelerate an unsupervised learning as the Kohonen or Self-Organizing Map (SOM) [14], [15] on a FPGA platform based on stochastic logic. In this context, stochastic logic is a candidate to reduce the power consumption, while maintaining low overall energy depending on the evaluation time. In addition, the tiny hardware resources requirements potentially allow for massively parallel implementations. Several complex functions design to implement the SOM Best Matching Unit (BMU) are presented and theoretically analyzed, as Winner-Take-All (WTA) module [16] and the Pseudo-Euclidian Distance (PED) between inputs and the neuron weights. The proposed approach is tested through a pattern recognition task, based on the Fisher's Iris flower data set.

The paper is organized as follows: Section II briefly introduces the basis of stochastic logic. Section III presents the stochastic logic basic blocks to implement the Kohonen map Best Matching Unit (BMU). Section IV shows and discuss a classification task results task performed with the stochastic BMU. Finally, the conclusions are presented in section V.

II. Stochastic Logic

In stochastic logic, a global clock provides a time interval during which each node of the circuit is stable. A probability p to be at high state can be defined during this interval. This probabilistic-based coding provides a natural way of operating with analog quantities using digital circuits and stochastic bit streams. The bit sequence follows probabilistic laws when they are evaluated through logic gates. This coding can represent

This work was supported in part by the Spanish Ministry of Economy and competitiveness and the EU with Regional European Development Founds under Grant TEC2017-84877-R.

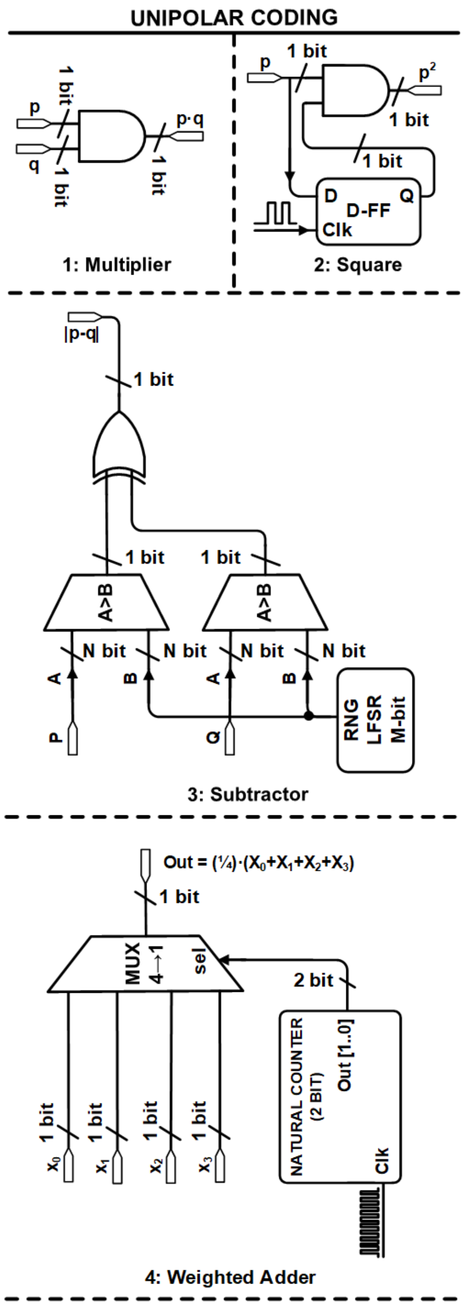


Fig. 1: Stochastic Logic basic circuits (Unipolar Coding)

only probabilities defined between 0 and 1, and is known as unipolar coding. For instance, an AND gate provides an output signal with a switching probability equal to the product of its inputs (the collision probability between signals) (Fig. 1.1). In turn, the square of a probability associated to a stochastic bit stream can be implemented with an AND gate but now we must include a delay to temporally de-correlate the bit stream to operate it with itself. This can be done using a delay chain of *D-FF* (Fig. 1.2). The subtraction of two unipolar signals (*p* and *q*) is not supported by this coding, since it cannot encode negative values; to circumvent this inconvenience is necessary to proceed from an alternative way, using the max-min algebra properties. The stochastic signals should be temporally correlated (sharing the same random number value) to evaluate their similarity, which is equivalent to determine the absolute value of the probabilities difference |*p* - *q*| [16]. This operation may be implemented with a XOR logic gate (Fig. 1.3). The addition of two (*u* and *v*) or more stochastic bit streams has been done historically with an OR logic gate that evaluate the sum of the probability of both signals minus the probability of collision between them, according to (1).

$$\begin{cases} out = u + v \rightarrow \mathbb{E}\big(OUT(U,V)\big) \\ \mathbb{E}(u+v) = p(U) + p(V) - p(U)\cdot p(V) = \cdots \\ = \frac{U}{2^n} + \frac{V}{2^n} - \frac{U\cdot V}{2^{2n}},\ n{:}\, Number\ of\ bits \\ \Downarrow \\ out = (\mathrm{Max}\, 1, u + v - u\cdot v) \end{cases} \tag{1}$$

But, OR based addition underestimates the sum for values of the entries by up to 50%, and the output saturates for values higher than the coding representation range [0, +1]. To overcome both limitations, a circuit that implements the weighted sum of the inputs with a multiplexer and a binary counter is proposed (Fig. 1.4). The multiplexer has 4 inputs (X_1 to X_4 through which the stochastic signals will be introduced) and a 2-bit selection word "*sel*" interconnected with a natural Up-counter, thus obtaining a linear combination of the inputs probability distributions, according to (2).

$$\begin{cases} out = x_1 + x_2 + x_3 + x_4 \rightarrow \mathbb{E}\big(OUT(X_1 + X_2 + X_3 + X_4)\big) \\ \mathbb{E}(x_1 + x_2 + x_3 + x_4) = \sum_{i=1}^{4} p(sel = i)\cdot x_i = \cdots \\ = \sum_{i=1}^{4} \frac{0.25\cdot X_i}{2^{2n}},\ n{:}\, Number\ of\ bits \end{cases} \tag{2}$$

In order to ensure that the sum is truly identically weighted (1/4), the selection signal must be chosen in a way that all the inputs signals are evaluated along the same number of cycles for a given integration period. For any stochastic coding, the main requirement to ensure the correct operation of these circuits is that the input bit streams must be uncorrelated.

A. *Stochastic Logic Architecture*

Stochastic logic system based is composed at least of three basic elements, as: A Binary-to-Pulse converter (*B2P*) to interface between the digital and the stochastic spaces, a stochastic circuit to carry out a certain task, and finally a Pulse-to-Binary converter (*P2B*) to newly interface the stochastic and digital spaces. An *N-bit* binary value can be converted to a stochastic bit-stream with a *B2P* block. This block is composed of a digital *N-bit* comparator that compares the digital input value with a uniformly distributed random number value, generated with a pseudo random number generator as a *M-bit* LFSR circuit. The LFSR has to meet the condition of *M-bit* ≥ *N-bit*. A stochastic bit-stream is converted to a N-bit binary value with a *P2B* block composed of a pulse counter of the stochastic bit-stream during *k* clock cycles (which corresponds to a system evaluation period $T_{eval} = k\cdot T$, where *T* is the clock period). The output is a binary number that remains fixed along *k* clock cycles until it is updated by the next bit-stream input integration value. This conversion incorporates a statistical error since the binary output value follows a binomial distribution, archiving a maximum conversion error of the order of $Error \approx k^{-0.5}$. Therefore exists a relationship between the evaluation period and the conversion error.

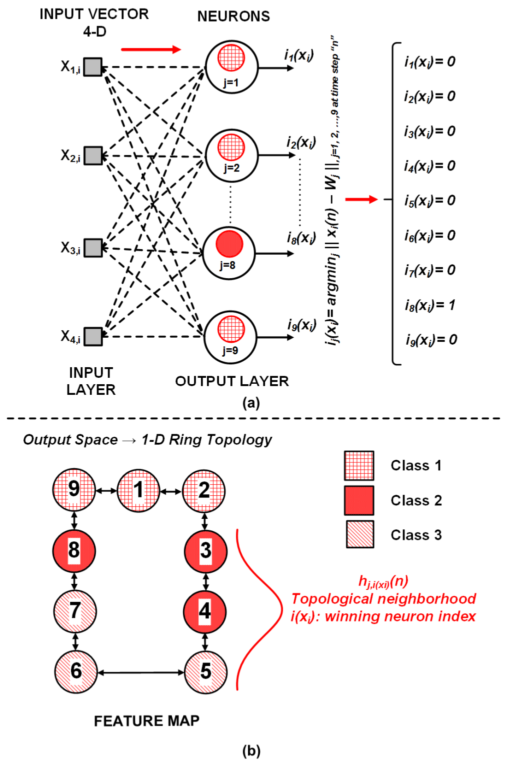


Fig. 2. (a) Kohonen Map. (b) 1D Lattice Ring Topology.

III. Kohonen Map Based on Stochastic Logic

This section briefly introduces the processes involved in the formation of the Kohonen feature map: competition, cooperation and synaptic adaptation. Below is presented the algorithm responsible for the formation of the self-organizing map, paying special interest in the description of the cooperation and synaptic adaptation processes algorithm coded in C++ on the embedded ARM Cortex-A9 processor in the Intel SoC-FPGA. Finally, the stochastic circuits designed to implement the competition process done by the Best Matching Unit (BMU) are presented.

A. Architecture

The Kohonen Map is a type of Artificial Neural Network (ANN) related to a feed-forward architecture composed by only two layers [14]. However, this type of architecture is fundamentally different in arrangement and motivation to the feed-forward ANN. These ANN applies Competitive and unsupervised learning method to train the network, based on grid of artificial neuron whose weights are adapted to match the input vectors in a training set. Specifically, Kohonen map approximate an unlimited number of inputs by a finite set of clusters/neurons arranged in a n-dimensional lattice (generally 1-D or 2-D), where the neighbor nodes correspond to more similar models. Kohonen map is mainly useful for clustering and visualization by creating a low dimensional feature map of high dimensional data sets.

The network input layer given a set of n input vectors $\{x_i \in \Re^k, i = 1, \ldots, n\}$ k-dimensional has k units. The output layer or the visible part of the Kohonen Map is the feature space, which consist of m nodes or neurons. The feature map space is defined beforehand, usually ranged in a n-dimensional region where nodes are arranged in a regular hexagonal or rectangular grid [17], but also its possible to use one-dimensional cyclic arrangement like in this work (Fig. 2.b). While the neurons positions in the feature map space remains fixed, the competitive training consists in moving neurons weight vectors toward the input data preserving the topology induced from the feature map space. Each neuron or node is associated with a weight vector w_j with the same dimension as each input vector $\{w_j \in \Re^k, j = 1, \ldots, m\}$. Therefore, the input units are fully connected and weighed with the output layer neurons. Thus, the output layer shows a feature map that describes a mapping from a higher-dimensional input space to a lower-dimensional map space. Once trained the neural network, the feature map can classify a vector from the input space by finding the neuron with the closest weight vector to the input space vector.

B. Self-Learning Algorithm

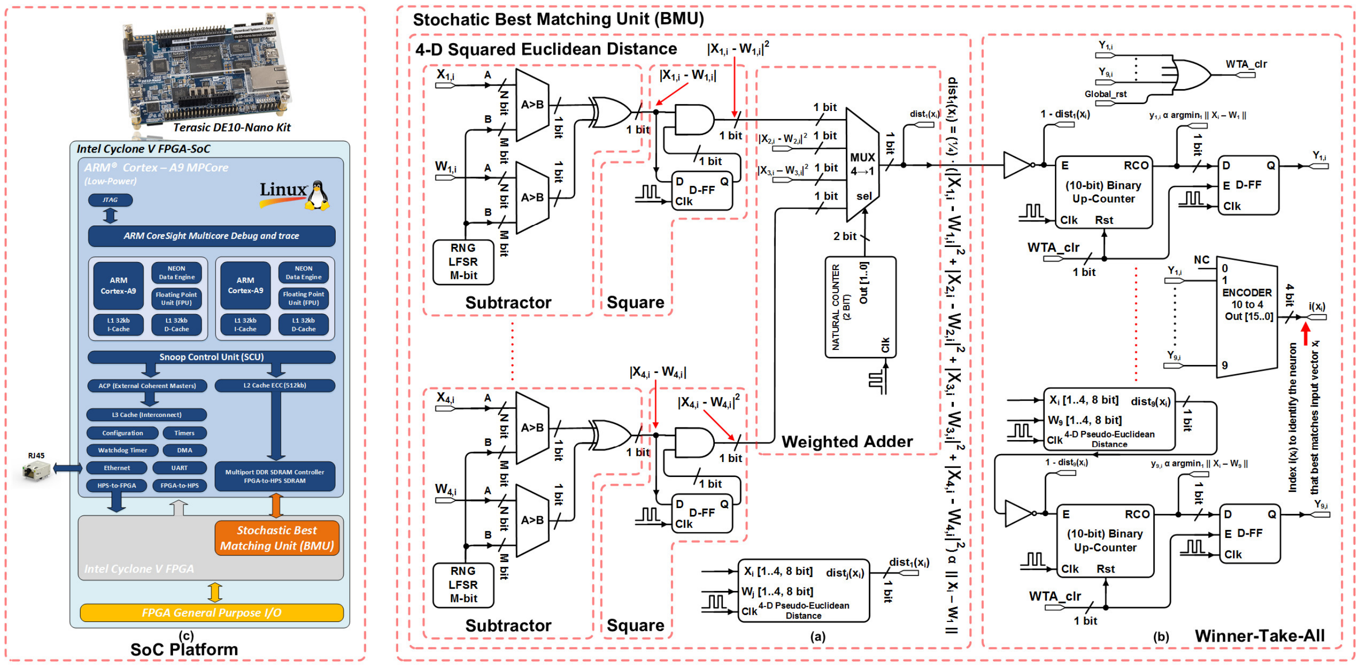
On Kohonen map, learning the weights neurons participates in a kind of competition for each input vector. Then, the winner of the competition and neighborhood neurons are allowed to change their weights following a Hebbian-Learning like rule [18]. There are two basic steps involved in the application of the self-learning algorithm after the output layer initialization, as: the similarity matching and weight updating. These two steps must be repeated until formation of the feature map has been completed.

The initialization step consists in choosing uniformly distributed random values for the initial weights vectors $W_j(0)$ of the output layer. The only restriction is related that the $W_j(0)$ be different for $j=1,2,\ldots,l$, where "l" is the number of neurons/clusters in the lattice. It may be desirable to keep the magnitude of the weights small, in the representation range selected [0, +1]. Another way to initializing the values on the output layer weights vectors is to select the weights vectors $\{w_j(0)\}_{j=1}^{l}$ from the dataset of input vectors $\{x_i\}_{i=1}^{N}$ in a random manner. In this work the weights initialization has been done in a random way.

The second step consists in evaluating the similarity matching to find the best-matching (winning) neuron index "$i(x_i)$" at time-step "t" by using the minimum-distance criterion. In this work the discriminant function defined has been the squared Euclidian Distance (ED) between the input vector $\{x_i\}_{i=1}^{N}$ and the weight vector $\{W_{j,i}(t)\}_{j=1}^{l}$ for each neuron. Therefore, the neuron whose weight vector comes closest to the input vector is declared the winner. So, this step provides the basic mechanism for competition among the neurons.

$$\begin{cases} i(x) = argmin_j \|x_i(t) - w_j(t)\|, & j = 1,2,\ldots,l \\ i(x) = argmin_j \left(\sum_{u=1}^{k} \left(x_i(t) - w_j(t)\right)^2\right), k\text{: input dim.} \end{cases} \quad (3)$$

The last step implements the synaptic-weights vector adaptive process in the self-organized formation of the feature map, in charge of adapting the output layer weights by using the update formula [14]:



Stochastic Best Matching Unit (BMU)
4-D Squared Euclidean Distance
Subtractor
Square
Weighted Adder
RNG LFSR M-bit
MUX 4→1
NATURAL COUNTER (2 BIT) Out [1..0]
ENCODER 10 to 4 Out [15..0]
(10-bit) Binary Up-Counter
D-FF
WTA_clr
Global_rst
4-D Pseudo-Euclidean Distance
Index i(x) to identify the neuron that best matches input vector x
(a)
(b)
Winner-Take-All
(c)
SoC Platform
Terasic DE10-Nano Kit
Intel Cyclone V FPGA-SoC
ARM® Cortex – A9 MPCore
ARM CoreSight Multicore Debug and trace
Snoop Control Unit (SCU)
Multiport DDR SDRAM Controller FPGA-to-HPS SDRAM
Stochastic Best Matching Unit (BMU)
Intel Cyclone V FPGA
FPGA General Purpose I/O
RJ45

Fig. 3. (a) 4-D Stochastic Squared Euclidean Distance Circuit. (b) Winner-Take-All Circuit. (c) SoC Platform Architecture.

$$w_j(t+1) = w_j(t) + \eta(n) \cdot h_{j,i(x)}(n) \cdot \left(x_i(t) - w_j(t)\right) \quad (4)$$

Where $\eta(n)$ is the learning-rate parameter and $h_{j,i(x)}(n)$ is the neighborhood function centered around the winning neuron index $i(x_i)$ which also incorporates the lattice topology and boundaries conditions. Both parameters are varied along the epoch dynamically during the learning for best results. In turn, neurobiological data suggest that the topological neighborhood should be symmetric and monotonically decreasing with the distance $d_{j,i}$ to the winning neuron lattice position. These requirements make a good choice the Gaussian function.

$$h_{j,i(x)}(n) = e^{-\frac{d_{j,i(x)}^2}{2\sigma^2(n)}},\ j = 1, \dots, l \quad (5)$$

In this work, the topology selected has been an 1D ring topology, $d_{j,i(x)} = |j - i(x)|/l$, due to its ease hardware synthesis and software calculation. Another feature of the SOM algorithm is that the size of the topological neighborhood decays exponentially with time/epoch "n" [19], described by:

$$\sigma(n) = \sigma_0 \cdot e^{-\frac{n}{\tau_1}}, n = 0,1, \dots, \#epoch \quad (6)$$

Where σ_0 is the value of $\sigma(0)$ at the initiation of the self-learning algorithm and $\tau_1$is a time constant to be chosen by the designer. In practice, the appropriate weight update equation must incorporate some kind of adaptive, time varying learning rate $\eta(n)$. In particular, it should start at some initial value η_0 and then decrease gradually with increasing epochs "n". This requirement can be satisfied by the following heuristic function:

$$\eta(n) = \eta_0 \cdot e^{-\frac{n}{\tau_2}}\ n = 0,1, \dots, \#epoch \quad (7)$$

Finally, the adaptive process could be repeated "n" epochs until no noticeable changes in the feature map are observed.

C. Best Matching Unit

Kohonen Map training is an iterative process through epoch which requires a lot of computational effort and thus is time-consuming. This training takes the input data-set vectors and infers them to the Kohonen map. The teaching consists of choosing a winner neuron/cluster by the means of a similarity measure and updating the values of the weights vectors in the neighborhood of the winner neuron. This process is repeated a large number of times until no changes in the feature map are observed. A detailed analysis shows how parts of the Kohonen map can be performed in parallel, to hardware accelerate the process as squared Euclidian distance calculation between the inputs vectors and the output layer neurons. To search for the Best Matching Unit (BMU) whose distance is the minimum, all distances are inevitably required to compare with each other. In this sub-section, the SoC architecture proposed is presented.

1) Stochastic Logic Implementation

A Best Matching Unit (BMU) is a computational block that evaluates the lattice index of the output layer neuron whose weight vector is most similar to an input vector, i.e. whose distance is the minimum, according to (3). Therefore, the search for the winning neuron index can be divided into two processes. One related with calculating similarity between the input vector and the whole of the output layer neuron weights, and the other in charge of selecting the winner neuron-index comparing all the calculated similarities. The process of searching a winner in a large lattice requires a long calculation time, because these processes conventionally are serially estimated.

The squared ED are one of the most popular ways to measure the distance or similarity between input vector $x_i(t)$ ant the lattice neuron weight vector $w_{j,i}(t)$. This Squared Euclidian Distance easily can be implemented stochastically using unipolar coding. To digitally implement the *k-dimension* squared ED, equation (3), calculation is necessary to combine a set of three basic stochastic blocks. The first one is composed by set of "k" unipolar subtractor blocks (as many as input vector dimensions) to evaluate the absolute value of the distance

TABLE I

KOHONEN MAP PARAMETERS AND WEIGHTS

	Output Layer Neuron Self-Trained Weights (w_j^k)			
Neuron$_j$/Dimension^k	w_j^1	w_j^2	w_j^3	w_j^4
1	0.835	0.380	0.585	0.177
2	0.953	0.378	0.825	0.248
3	0.836	0.369	0.697	0.255
4	0.756	0.340	0.628	0.213
5	0.721	0.330	0.536	0.154
6	0.662	0.313	0.416	0.126
7	0.628	0.368	0.267	0.055
8	0.591	0.390	0.172	0.014
9	0.654	0.453	0.183	0.023
Kohonen Map initialization parameters				
σ_0	0.5	τ_1	100	
η_0	0.5	τ_2	100	
BMU counter modulus "k"	1,024	**# epoch**	100	
MATLAB® Classification Performance				
Test Set		**Archived Classification Error**	**Classification Error [%]**	
75		1	1.3	

TABLE II

STOCHASTIC BMU MODULUS IN FRONT OF MISCLASSIFICATIONS

BMU Counter Modulus "k"	Test-Set Number of Misclassifications	Test-Set Misclassifications [%]
128	46	61.3
256	24	32.0
512	14	18.7
1024	0	0

TABLE III

FPGA RESOURCES (CYCLONE V, 5CSEBA6U23I7NDK)

Adaptive Logic Module (ALM):	**614** / 41,910 (1,46%)
Total Thermal Power Dissipation:	**21.5 mW**

between the input vector $x_i(t)^k$ and the neuron weight $w_{j,i}(t)^k$ for each dimension component $\left|x_i(t)^k - w_{j,i}(t)^k\right|$, as depicted in Fig. 3a for the 4-D input vector case. Next, the subtractions must be squared using an AND gate to multiply the distance stochastic bit stream by itself delayed six clock cycles. Finally, each squared dimension distance must be added to obtain the squared ED. In this case the stochastic adder only can evaluate the weighted sum of the *k* distances instead of the full sum. But the target is to determine the similarity between vectors and therefore independently how the addition is implemented the most similar vector will be whose distance is smaller. The Winner-Take-All (WTA) circuit, Fig. 3b, given a set of "*l*" inputs (as many as lattice neurons) determines which input has associated the minimum number of high level values in an evaluation period. The WTA stochastic design takes each neuron distance "$d_j(x_i)$" stochastic signal and through a Not gate evaluates the complementary signal probability "$1 - d_j(x_i)$" in order to associate to the minimum distance the bit-stream with the highest number of ones. Then these distance complementary signals attacks "*l*" binary counters (*module-k*). In turn, only a maximum number of "*z*" clock cycles per comparison are allowed (*z*>*k*), and therefore the minimum number of cycles necessary needed to overflow a WTA counter fixes a minimum distance value to be distinguished "d_{min}", so that $d_{min} = k/z$. Therefore, for any distance value $d_j(x_i) > d_{min}$ between an input x_i and the output layer weights $w_j(t)$ the probability to identify this neuron "*j*" as winner is close to "1". In this work the value of k has set to 1024 and *z* to 4096. When one of the counters overflows, its ripple carry-out signal resets the counters and activate a flag (through a D-FF) indicating the winning neuron. Finally, through an encoder, an unsigned binary number containing the index of the winning neuron is generated.

2) *System-on-Chip FPGA*

An overview of the System-on-Chip FPGA for Kohonen Map self-learning acceleration based on Stochastic logic for IoT applications is presented in Fig. 3c. The proposed hardware is coded in VHDL and synthesized with Intel Corp. Quartus Prime 18.1 software, and implemented on a low cost Terasic DE10-Nano-SoC FPGA educative board, equipped with a Cyclone V FPGA model 5CSEBA6U23I7. The proposed hardware in this work consist of 9 circuits that performs 4-dimensional squared pseudo Euclidean distance between the input vector and weight vectors, both coded with 8-bit unsigned integer; and this circuit operates at 50MHz. The Kohonen map application on the dual core ARM Cortex-A9 embedded on the Cyclone V is coded in C++ and runs on Ubuntu Linux 16.04. This application is responsible for loading an external file with a data set, but also can take as input the output of some sensors, and is responsible for randomly cut the data set in two parts (training-set and test-set). Then it randomly initializes the weights of the output layer neurons completing the initialization of the self-learning algorithm. Next, the application sends to the FPGA the weights of the 9 neurons and an input vector *X$_i$* through a dual-port FIFO implemented on Multiport DDR SDRAM Controller that allows the communication between the Hard Processor System (HPS) and the FPGA. So that once the winning category is evaluated, the FPGA returns the winning neuron index *i(x)* to the HPS, in a maximum of 82µs, using a new Memory-Mapped FIFO. Once the embedded C++ application receives the winning index, its adapts the output weights, following the procedure described in the section III. Finally, the application repeats this procedure for all the vectors of the training set up to 100 epochs; at which time the variations in the output weights are imperceptible. To later proceed to evaluate the result of self-learning by classifying the vectors of the test-set.

IV. RESULTS

The Fisher's Iris is a multivariate data set widely used to test machine-learning algorithms. The data set consists of 50 samples from each of three Iris flower species (Setosa, Virginica and Versicolor). Each sample has four features: the length and the width of the sepals and petals, in millimeters. This dataset is used to perform a pattern recognition task, in order to compare the software-based model (MATLAB®) and the proposed SoC Kohonen map acceleration system results. The dataset is divided randomly into two identical parts, the first for training (75 vectors) and the second for testing (75 vectors). To carry out this task, SoC acceleration system has been implemented and tested on a FPGA. This SoC Kohonen map accelerator incorporates 9 neurons located on 1D lattice in ring topology for 4D inputs as presented in section III. Table I presents the self-learning initial parameters ($\sigma_0, \tau_1, \eta_0, \tau_2$) and the self-trained weights values

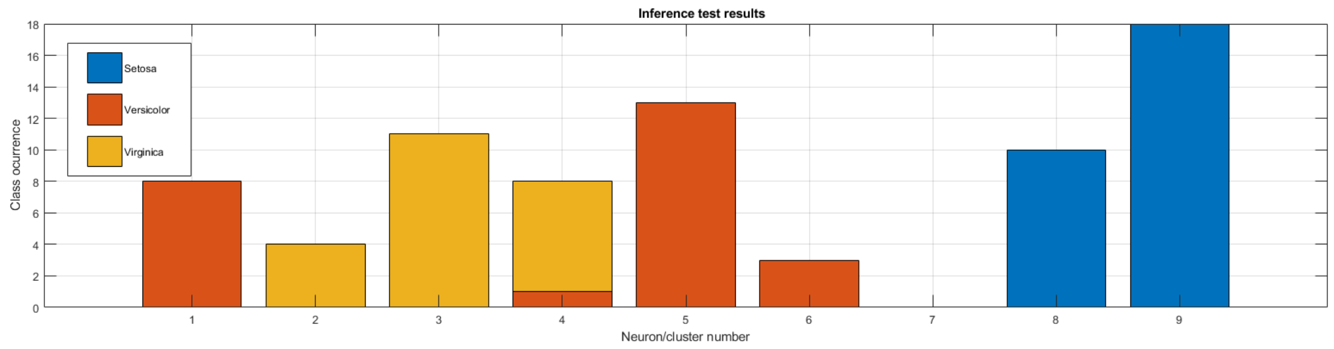


Fig. 4: SoC FPGA Fisher's Iris classification task results ($k = 1024$); the height of the bars indicates the number of times a given node is the winner and different colors represent different output classes (Iris species), so that overlaps represent misclassifications.

obtained after 100 epochs along with the MATLAB® application test set classification error. The classification task results for each neuron carried on the SoC FPGA are presented in the Fig. 4, with WTA counters modulus 1024 and an evaluation period of 4096 clock cycles or 82µs. The SoC implementation output are depicted with a bar graph (color are related with Iris species). As can be appreciated, the experimental classification results with this configuration are identical to the obtained with the reference software implementation, presenting a classification error of only the 1.33% or 1 misclassification for 75 test-set vectors. Additionally, in Table II is presented the relationship between the BMU counter modulus and the number of test set vector misclassified or unclassified respect the purely Matlab® Kohonen Map results, for the presented in Table I self-trained weights values. Finally, the FPGA hardware resources spent in this implementation without considering the interface blocks (Memory interface, Finite State Machine (FSM), …) are presented in Table III.

V. Conclusions

The SoC FPGA architecture based on stochastic computing have been presented and evaluated. Also, a set of stochastic circuits has been described to efficiently implement self-learning tasks in hardware. The results show for the SoC Kohonen map acceleration solution can perform the self-learning and classification tasks with the same error as MATLAB® by an integration period of $k = 1024$ cycles, all using simple digital blocks. Therefore, the solution proposed can control the classification accuracy of the system based on the integration period of the BMU block. In turn, the stochastic BMU hardware accelerator circuit consumes 21.5 mW, i.e. around 4 times less power consumption than the IoT devices (typically 80-86 mW) [20] in the sensing or computation phases, which makes it suitable for use in these applications. Also, this implementation consumes few hardware resources (614 ALM in a FPGA). In addition, the stochastic BMU is able to work properly with a high level of noise at the inputs due to its stochastic nature.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep Learning for IoT Big Data and Streaming Analytics: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[3] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, and M. Villari, "From the Cloud to Edge and IoT: a Smart Orchestration Architecture for Enabling Osmotic Computing," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2018, pp. 419–424.

[4] J. G. Tong, I. D. L. Anderson, and M. A. S. Khalid, "Soft-Core Processors for Embedded Systems," in *2006 International Conference on Microelectronics*, 2006, pp. 170–173.

[5] J. Cong, Bin Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Zhiru Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.

[6] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. New York, NY, USA: Cambridge University Press, 2014.

[7] S. Ray, "A Quick Review of Machine Learning Algorithms," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 35–39.

[8] Xue-Wen Chen and Xiaotong Lin, "Big Data Deep Learning: Challenges and Perspectives," *IEEE Access*, vol. 2, pp. 514–525, 2014.

[9] R. W. Keyes, "Fundamental limits of silicon technology," *Proc. IEEE*, vol. 89, no. 3, pp. 227–239, Mar. 2001.

[10] T. Moreau *et al.*, "A Taxonomy of General Purpose Approximate Computing Techniques," *IEEE Embed. Syst. Lett.*, pp. 1–1, 2017.

[11] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rossello, "A New Stochastic Computing Methodology for Efficient Neural Network Implementation," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 27, no. 3, pp. 551–564, Mar. 2016.

[12] V. Canals *et al.*, "Noise tolerant probabilistic logic for statistical pattern recognition applications," *Integr. Comput. Aided. Eng.*, vol. 24, no. 4, pp. 351–365, Sep. 2017.

[13] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, May 2013.

[14] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.

[15] N. R. Pal, J. C. Bezdek, and E. C.-K. Tsao, "Generalized clustering networks and Kohonen's self-organizing scheme," *IEEE Trans. Neural Networks*, vol. 4, no. 4, pp. 549–557, Jul. 1993.

[16] A. Morro, V. Canals, A. Oliver, M. L. Alomar, and J. L. Rossello, "Ultra-Fast Data-Mining Hardware Architecture Based on Stochastic Computing," *PLoS One*, vol. 10, no. 5, p. e0124176, May 2015.

[17] Kangas, Kohonen, Laaksonen, Simula, and Venta, "Variants of self-

organizing maps," in *International Joint Conference on Neural Networks*, 1989, pp. 517–522 vol.2.

[18] R. H. White, "Competitive Hebbian learning," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. ii, p. 949.

[19] K. Obermayer, K. Schulten, and G. G. Blasdel, "A Comparison Between a Neural Network Model for the Formation of Brain Maps and Experimental Data," in *Proceedings of the 4th International Conference on Neural Information Processing Systems*, 1991, pp. 83–90.

[20] N. Tamkittikhun, A. Hussain, and F. A. Kraemer, "Energy Consumption Estimation for Energy-Aware, Adaptive Sensing Applications," in *Mobile, Secure, and Programmable Networking. MSPN 2017. Lecture Notes in Computer Science, vol 10566*, Springer, Cham, 2017, pp. 222–235.