Exploiting Correlation in Stochastic Computing Based Deep Neural Networks

Christiam Camilo Franco Frasser, Pablo Linares, Alejandro Morán, Joan Font-Rosselló, Vincens Canals, Miquel Roca, Teresa Serrano Gotarredona and Josep Lluis Rosselló

September 4, 2021

Exploiting Correlation in Stochastic Computing based Deep Neural Networks

***Abstract*—A new trans-disciplinary knowledge area, Edge Artificial Intelligence or Edge Intelligence, is beginning to receive a tremendous amount of interest. Unfortunately, the incorporation of AI characteristics to edge computing devices presents the drawbacks of being power and area hungry for typical machine learning techniques such as Convolutional Neural Networks (CNN). In this work, we propose a new power-and-area-efficient architecture for implementing Artificial Neural Networks (ANNs) in hardware, based on the exploitation of correlation phenomenon in Stochastic Computing (SC) systems. The architecture purposed can solve the difficult implementation challenges that SC presents for CNN applications, such as the high resources used in binary-to-stochastic conversion, the inaccuracy produced by undesired correlation between signals, and the stochastic maximum function implementation. Compared with traditional binary logic implementations, experimental results showed an improvement of 19.6x and 6.3x in terms of speed performance and energy efficiency, for the FPGA implementation. For the first time, a fully-parallel CNN as LENET-5 is embedded and tested in a single FPGA, showing the benefits of using stochastic computing for embedded applications, in contrast to traditional binary logic implementations.**

***Index Terms*—Stochastic Computing, Edge computing, Convolutional Neural Networks.**

EDGE computing (EC) is characterized by the implementation of data processing at the edge of the network [1] instead of at the server level. This has produced great interest in the Microelectronic industry due to the proliferation of the Internet of Things (IoT). At the same time, incorporating Artificial Intelligence (AI) capacities in everyday devices has been in the spotlight in recent times, and it continues to be a hot topic, making the development of new techniques to extend AI to edge applications a must [2]. The idea behind these research efforts is to assist EC devices to further reduce their dependence on cloud processing by considerably reducing the energy associated with data transmission considering only the relevant information exchange with the cloud server. However, research on Edge Intelligence is still in its early days, since edge nodes normally present considerable limits in terms of area and power consumption, producing an intrinsic complexity for typical state-of-the-art deep learning implementations in embedded devices. That is why new solutions for efficient hardware implementations for machine learning applications such as Convolutional Neural Networks (CNNs) have become a trending topic.

Stochastic Computing (SC), developed during the sixties [3] as an alternative to traditional binary logic, is an approximate computing technique that has been arousing increasing interest over the last decade thanks to its capacity to compress complex functions within a low number of logic gates. Such characteristic has motivated the development of different proposals for the use of SC to implement ANNs in hardware [4], [5], [6], [7], [8], [9], and more specifically to implement CNNs [4], [5], [10], [11]. However, SC implementations face their own challenges such as: (a) the cost in terms of hardware resources required to implement different Random Number Generators (RNG), (b) the precision degradation between layers produced by the lack of full decorrelation between signals, and (c) the implementation of an efficient stochastic maximum function. Tackling these issues is not trivial, Lee et al. [12] has approached them by implementing only the first convolutional layer using stochastic computing. Sim et al. [13] has created an hybrid stochastic-binary architecture, where only the multiplications are implemented in SC. Both approaches are not fully-stochastic, and therefore the benefits are limited.

In this work, we propose an efficient and compact hardware architecture to deal with these hurdles by exploiting the correlation and decorrelation between SC signals in such a way as to implement the CNN basic building blocks. As a proof of concept, we implement a fully-parallel SC CNN in a single FPGA chip and compare its performance with different CNN FPGA implementations using traditional binary logic.

I. Stochastic Computing

A. Unipolar and bipolar codification

Stochastic computing (SC) is an approximate computing methodology that represents signals using the switching frequency of time-dependent bit-streams. The SC signal is composed of pulses that represent the probability of finding a TRUE value (logic '1') at any arbitrary position throughout the sequence of bits. For instance, the number 0.75 could be represented by a bit-stream in which the probability of finding a logic '1' along the bit-stream is 75%: (1,1,0,1) for a four bit-stream or (0,1,1,0,1,1,1,1) for an eight bit-stream. Representing only positive values (between 0 and 1) is known as *unipolar* codification. To represent negative values, a different codification is required: the *bipolar* codification. In *bipolar* codification the number of zeros is subtracted from the number of ones, and finally divided by the total number of bits in the stream: $p^* = (N_1 - N_0)/(N_0 + N_1)$, where N_0 and N_1 are the number of zeros and ones respectively, and the * symbol denotes *bipolar* codification. This expression is equivalent to implementing a change of variable: $p^* = 2p - 1$, where *p* is the *unipolar* representation of the number. As noted, the *bipolar* codification provides a range of possible values in the interval $[-1, 1]$.

In the SC paradigm, each magnitude X is converted to its time-dependent stochastic counterpart $x(t)$ by using a random number generator $R(t)$ and a comparator, so that the stochastic signal may be understood as a sequence of booleans $x(t) = \{X > R(t)\}$. If the number X is greater than the random number $R(t)$, the output is set to '1' (assigned to the TRUE value of the comparison), otherwise it is set to '0' (FALSE value). If the random variable generated by $R(t)$ is uniform in the interval of all possible values of X, the mean switching probability $\bar{x}$ of the stochastic signal $x(t)$ is proportional to the converted magnitude X. In order to recover the X value, a digital counter is incremented every high pulse of the bit-stream during a fixed period of time. The time length over which the sum is performed is related to the conversion error, so that the longer the time, the lower the error.

One of the main advantages of using SC is the low cost in hardware resources of implementing complex functions. Take for instance the multiplication operation, which is implemented in SC using just a single logic gate: an AND gate for *unipolar* codification and an XNOR gate for *bipolar* codification. Fig. 1 shows how the same arithmetic operation could be achieved using different logic gates for different codification techniques in the presence of the same input waves.

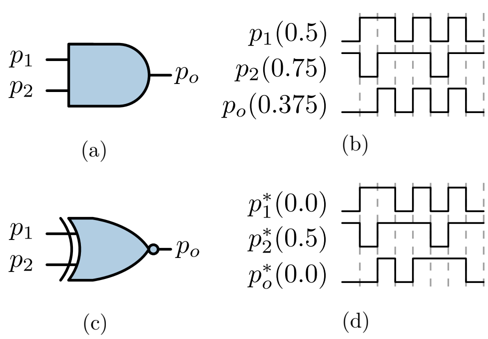

Fig. 1. Stochastic multiplication using different codification techniques with the same input bit-streams: (a) *unipolar* multiplication gate, (b) time diagram for *unipolar* multiplication circuit, (c) *bipolar* multiplication gate, (d) time diagram for *bipolar* multiplication circuit.

B. Stochastic correlation

In order to generate stochastic bit-streams $x(t)$ from value X, a converter circuit must be implemented. The most commonly used circuit is based on a pseudo-random number generator, normally a Linear Feedback Shift Register (LFSR) and a comparator. The converter circuit is noted as BSC (Binary-to-Stochastic-Converter) in Fig. 2, where for the upper block the X represents the signal to be converted, $R(t)$ the random value provided by the LFSR, and $x(t)$ the stochastic bit-stream generated.

Two bit-streams are said to be correlated when both have some statistical similarities as discussed in [14]. To produce the maximum correlation between two bit-streams, we can connect the same LFSR output $R(t)$ to the reference input of both comparators when performing the BSC conversion. A different pseudo-random number generator $R'(t)$ is used in case decorrelation is desired to operate the stochastic signals, producing a different outcome.

To quantify the correlation, we can use the independence factor defined in [15] or its dual, the stochastic computing correlation factor:

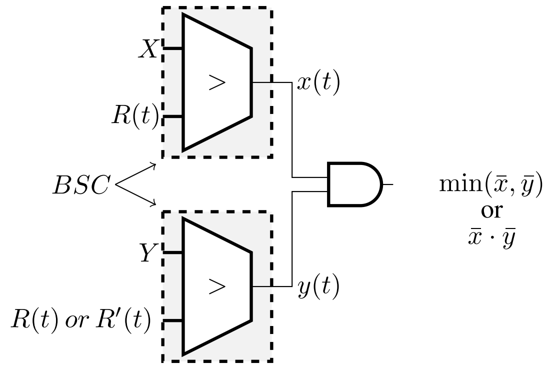

Fig. 2. Correlation impact over stochastic operations. Correlation between signals changes the operation computed by the logic gate. Stochastic signals $x(t)$ and $y(t)$ are said to be totally correlated when they share the same random number generator $R(t)$, producing the function $\min(\bar{x}, \bar{y})$; otherwise, if $R'(t)$ is connected, they are said to be decorrelated and the output function is different: $\bar{x} \cdot \bar{y}$.

$$C(x(t), y(t)) = \frac{Cov(x(t), y(t))}{\min(\bar{x}, \bar{y}) - \bar{x}\bar{y}} \tag{1}$$

where function Cov is the covariance between the two time-dependent stochastic signals $x(t)$ and $y(t)$, while parameters $\bar{x}$, $\bar{y}$ are their mean values (their probability of being '1'). A correlation value of $+1$ implies maximum probabilistic similarity, obtained by sharing the same random number generator; whereas a 0 value implies a complete decorrelation, produced by connecting two independent RNG as input reference comparison.

The stochastic output of a two-input gate can be expressed as a function of the correlation between its inputs. For the case of the AND and OR gates we have:

$$\begin{aligned} AND(x,y) = &\quad \bar{x}\bar{y} + \big(\min(\bar{x},\bar{y}) - \bar{x}\bar{y}\big)C(x,y) \\ OR(x,y) = &\quad \bar{x} + \bar{y} - \bar{x}\bar{y} + \big(\bar{x}\bar{y} - \min(\bar{x},\bar{y})\big)C(x,y) \end{aligned} \tag{2}$$

where as noted, the arithmetic operation is altered by the correlation level between the stochastic inputs.

Most of the errors produced by SC systems come from operating two stochastic signals with an undesired degree of correlation between them. Different efforts in literature have tried to operate with fully-uncorrelated stochastic signals in order to avoid the stochastic correlated imprecision. The main approach is done by generating all the aleatory $R(t)$ signals with independent LFSRs, thus, employing a high amount of hardware resources in the conversion circuits, and therefore, limiting the contributions that SC offers for hardware implementations. But despite the fact that decorrelation is necessary for many operations, there are some cases where correlated signals may be preferable. Consider the case of the AND gate of Fig. 2. In presence of two decorrelated signals (produced by using $R(t)$ and $R'(t)$ on each BSC), performs the multiplication operation: $\bar{x}\cdot\bar{y}$; while the *minimum* operation $\min(\bar{x}, \bar{y})$ is performed if the stochastic inputs are totally correlated (produced by sharing the same random generator $R(t)$ in the conversion circuit). This interesting feature can be exploited to produce high performance architectures that reduce area and power.

C. Stochastic addition

Due to boundaries of stochastic bit-stream representation, accurate implementations for the stochastic addition keeps being a challenge. Different circuits have been proposed to approximate the addition: a simple OR gate, a multiplexer, and an Accumulative Parallel Counter (APC). Fig. 3 shows the different stochastic addition circuits, where for the sake of clarity, and from now on, the stochastic signals are denoted without the time-dependant reference (t).

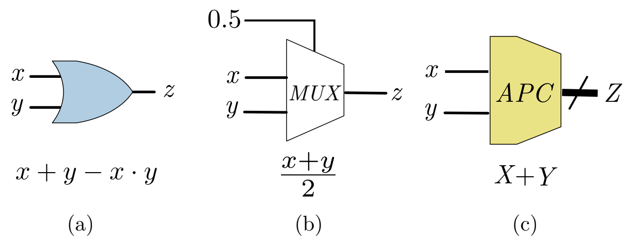

Fig. 3. Stochastic addition circuits: (a) Stochastic addition using an OR gate, where $x \cdot y$ must be close to zero in order to compute the addition accurately. (b) Stochastic scaled addition using a multiplexer, where the accuracy outcome is dependant of the number of inputs. (c) Stochastic addition using an Accumulative Parallel Counter (APC), where the accuracy has no degradation and the output is represented in binary format.

The OR gate is the smaller circuit in terms of hardware footprint, but it has the drawback of high inaccuracy outcomes when the input values are not close-to-zero. Not to mention its correlation dependant feature, discarding its use as an stochastic addition circuit in most applications.

The multiplexer is one of the most popular circuits to achieve the addition. The circuit is low-cost in terms of area and the precision is not affected by the correlation among the inputs. The main disadvantage is the inaccuracy increment as the number of inputs grow, being not suitable for deep learning implementations, where the number of inputs for the addition operation are high demanding.

The last case is the APC, which counts the number of the input high pulses and accumulates the counted value for a period of time, producing a complement-2 output (*bipolar* format). The APC solution is the most accurate from the circuits presented. Additionally, correlation among input signals does not disturb the result. The use of APC is the preferable approach for high precision implementations in spite of the higher resource utilization produced.

D. The Stochastic Neuron

Convolutional Neural Networks (CNNs) are constructed of several interconnected layers of neurons. The core neuron employed is composed of a scalar product block and a Rectified Linear Unit (ReLU) transfer function, which implements the operation: $\max(0, input)$. The common base operations used in CNN implementations are: the multiplication, the addition and the maximum function, employed for the max-pooling operation and the ReLU transfer function. They can be easily implemented in stochastic computing systems if correlation is properly used. In literature, different stochastic neuron designs have been proposed [4], [5], [6], [7], although non of them have exploited the signal correlation properties, which can simplify the CNN hardware in a considerable way.

Fig. 4 shows the proposed stochastic neuron design with the correlation exploiting architecture. The incoming stochastic vector $\mathbf{x}^*$ (composed of n elements) is generated using the output of one LFSR circuit $R_x(t)$, whereas the stochastic weight vector $\mathbf{w_i^*}$ (where i represents the i^{th} neuron of the current layer) is generated using the output of a second LFSR circuit $R_w(t)$ (not shown in the diagram); therefore, producing decorrelation between them. As a result, and considering a *bipolar* codification, the n-XNOR-gate array calculates the stochastic product between neuron inputs and weights.

Considering the APC is adding all the incoming signal products into a single complement-2 number, we can take advantage to connect the same pseudo-random number $R_x(t)$ used to generate a zero-*bipolar* (0^*) reference signal, to the BSC block that re-converts the APC outcome to the stochastic domain, thus producing fully correlation between both stochastic signals. Once in the stochastic domain, the ReLU activation function is easily implemented by using an OR gate, returning the operation: $y_i^* = \max(0^*, \sum_{j=1}^{n} x_j^* \cdot w_{ij}^*)$.

Since the same procedure is followed in all of the neurons, two unique pseudo-random number generators ($R_x(t)$ and $R_w(t)$) are needed to accomplish the whole calculus, considerably saving area and power in the design.

One of the benefits of the proposed stochastic-ReLU-function approach, is its normalized reproduction of the standard-ReLU-function used by the machine learning community. This means that the weights obtained after the training process of the ANN, can be straight-forward adapted to the hardware, since the expected activation function is not disturbing. This is in contrast with other published studies, in which the function outcome is distorted, as is the case of references [5], [11]; where the stochastic implementation of the ReLU function, besides being large area-consuming, is clipped and not exact. In our simple ReLU proposal, the OR gate implementation computes the maximum function without clipping or distorting the signal, and therefore, the weights of any standard training process considering ReLU-dependent neurons can be incorporated directly to the hardware after a simple process of normalization.

E. Max-pooling layer

In traditional CNNs, after convolutional layers extract the features from the input, a sub-sampling operation (pooling) is accomplished to reduce the spatial dimensions of the convoluted feature. In the case of Max-Pooling (MP), the sub-sampling is performed selecting the maximum value from a spatial window of the convoluted feature: $\max_{j=1}^{k}(y_j^*)$, where k is the size of the spatial window.

Ren et al.[4], Z.Li et al.[5], and Yu et al.[11] have proposed some stochastic maximum function designs using a set of counters, comparators and multiplexers. The drawbacks of these architectures are the employment of large area in hardware resources; and moreover, the designs they proposed only find out the maximum value after counting the total number of high pulses in the bit-stream for a period of time, incurring in

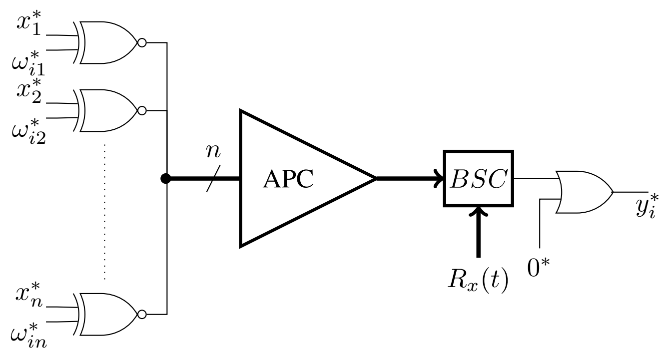

Fig. 4. Stochastic neuron design exploiting correlation to reduce area cost. Stochastic signals from *BSC* block and zero-*bipolar* (0^*) are generated with the same LFSR ($R_x(t)$) to produce total correlation between them, returning the maximum function on the output with a single OR gate.

long latency and considerable energy consumption. In contrast, our architecture takes advantage of the full correlation among the neuron outputs signals, and as in the ReLU transfer function case, it extracts the instantaneous maximum value with a unique OR gate (Fig. 5), saving precious area resources, latency time and energy consumption. For the case of Min-pooling or average pooling, the OR gate must be changed by an AND gate or a multiplexer respectively.

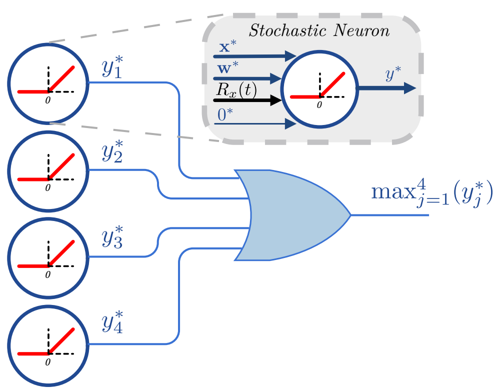

Fig. 5. Stochastic max-pooling circuit for a spatial window size of k = 2x2. Stochastic neuron outputs y_k^* are totally correlated, allowing the implementation of the maximum function with a single OR gate.

F. Full CNN Architecture

Figure 6 shows how the whole system is connected in the CNN. As shown, only two unique pseudo-random number generators ($R_x(t)$ and $R_w(t)$) are needed to accomplish the whole calculus, considerably saving area and power in the design. This could be achieved thanks to the stochastic neuron design, which exploits correlation and de-correlation for computing. LFSR1 is used for the $R_x(t)$ number generation, and is connected to the input image BSC conversion, the 0^* signal reference generator, and to every stochastic neuron in the whole design (for the APC stochastic generator, see Fig 4). LFSR2 is used for the $R_w(t)$ number generation, which is only used to produce the stochastic weights. In this way, each stochastic signal generated by LFSR1 is totally uncorrelated with those generated by LFSR2, allowing neuron inputs to be multiplied by weights with the highest precision. Moreover, the architecture proposed allows neuron outputs from layer l_i to be connected to the neuron inputs of the next layer l_{i+1} without any risk of signal degradation. Since the l_i neuron outputs are generated from a first LFSR block $R_x(t)$, and the l_{i+1} weights are generated from a second LFSR $R_w(t)$, the error induced from layer to layer by the appearance of uncontrolled correlation between signals is totally avoided.

As noted by dashed lines, $R_x(t)$ and 0^* are shared through the whole network, saving plenty of resources and allowing all neurons work simultaneously in parallel. Power consumption is saved dramatically as no access to memory for reading or writing intermediate results is accomplished.

II. Experimental Results

In order to evaluate the proposed stochastic design, we have implemented the LeNet-5 CNN [16]. This CNN design is oriented to processing the MNIST data set composed of 60k training images and 10k testing images. The CNN architecture consists of two convolutional layers and three fully connected layers, as the original paper describes [16] . The baseline score of the trained model, using floating point, was 98.6% (no special optimizations were introduced). On the other hand, the stochastic implementation score was 97.6%, only a 1% accuracy degradation compared to the software version; a satisfactory result, considering no parameter fine tuning process was applied, just a simple weight normalization. It is important to note that no prunning, weight sharing or clustering has been carried out in our architecture. The whole array of weights has been embedded in the design.

We tested the full SC CNN implementation in a GIDEL PROC10A board, which has an Intel 10AX115H3F34I2SG FPGA running the 8-bit SC implementation at 150MHz. The communications were done through PCI express bus.

Table I shows the comparison between the proposed SC implementation and conventional FPGA-based CNN accelerators presented in literature. The comparison has been made in terms of execution time latency, inference per second, and energy efficiency. As can be appreciated, the proposed method outperforms others architectures. Results show that the proposed stochastic CNN implementation achieves 19.6x more speed performance (measured in inferences per second and per megahertz) compared to the VX690T implementation [17], and a 6.3x more energy efficiency compared to Virtex7-485t implementation [18] (measured in inferences per Joule).

To the best of our knowledge, this is the first time an entire fully-parallel SC CNN is embedded into a single FPGA. This fact is in contrast to the studies presented in [19], [17], [18], [20], where the inference operations are accomplished using a loop-tiling technique (an optimization approach to use the same hardware resources recursively).

In our design, DSPs blocks are avoided, since an unconventional computing technique (Stochastic Computing) is used instead of traditional binary logic. At the same time, memory blocks are not required, since the computation is not performed in a tile-loop manner, thus, reducing the main power consumption source, which comes from the access operations to the memory.

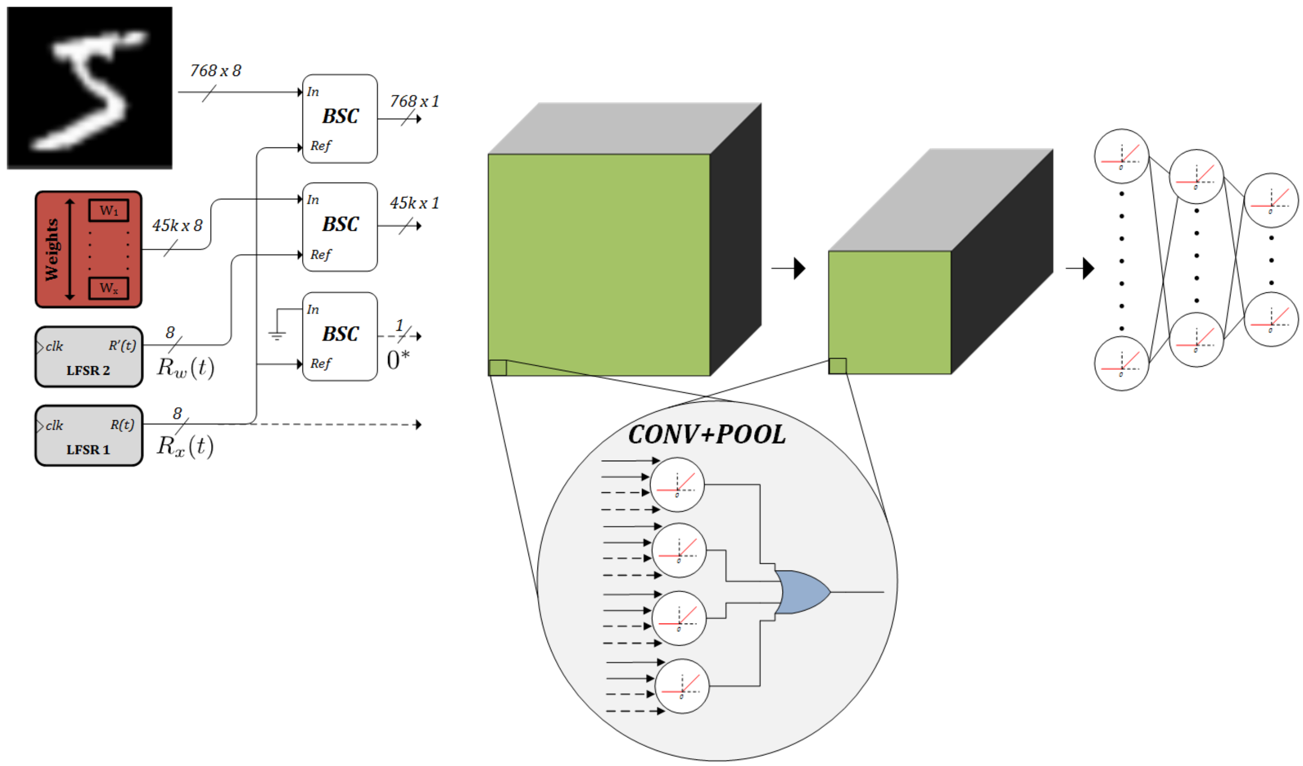

Fig. 6. Fully-parallel stochastic CNN architecture. Only two unique pseudo-random number generators are employed. All neurons are working simultaneously in parallel thanks to the correlation phenomenon exploitation.

In order to compare the area efficiency, the hardware area used by the FPGA specific blocks (DSP and memory) need to be known, but they are company-reserved; hence, we provide only the area efficiency value for our design (in terms of inferences per MHz and per logic unit ALM).

III. Conclusion

Thanks to the advantages of the small area and low power consumption, stochastic computing is presented as a paradigm solution to implement machine learning algorithms in hardware for edge computing. However, many difficulties are still being faced in the quest to achieve good results. In this paper, we present an efficient reduced-area architecture to deal with the high area consumed by random number generators, the precision degradation produced by correlation between signals, and the stochastic maximum function implementation. For the first time, a fully-parallel convolutional neural network is embedded in a single FPGA chip, obtaining better performance results compared to traditional binary logic implementations, showing the compression effectiveness of the architecture by exploiting the correlation features presented by stochastic signals.

Future work will be to synthesize the proposed architecture in VLSI and compare it with state-of-the-art chips employed as a solution for deep learning applications.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, 2019.

[3] B. R. Gaines and Others, "Stochastic computing systems," *Advances in information systems science*, vol. 2, no. 2, pp. 37–172, 1969.

[4] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 405–418, 2017.

[5] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu, and Y. Wang, "HEIF: Highly efficient stochastic computing-based inference framework for deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1543–1556, Aug 2019.

[6] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselló, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 551–564, March 2016.

[7] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, "Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function," *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 97–104, 2017.

[8] J. Rosselló, V. Canals, and A. Morro, "Hardware implementation of stochastic-based neural networks," in *Proceedings of the International Joint Conference on Neural Networks*, 2010.

[9] ——, "Probabilistic-based neural network implementation," in *Proceedings of the International Joint Conference on Neural Networks*, 2012.

[10] H. Sim and J. Lee, "Cost-effective stochastic mac circuits for deep neural networks," *Neural Networks*, vol. 117, pp. 152–162, 2019.

[11] J. Yu, K. Kim, J. Lee, and K. Choi, "Accurate and efficient stochastic computing hardware for convolutional neural networks," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 105–112.

[12] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 13–18.

[13] H. Sim, D. Nguyen, J. Lee, and K. Choi, "Scalable stochastic-computing accelerator for convolutional neural networks," in *2017 22nd Asia and*

TABLE I
COMPARISON WITH OTHER FPGA LENET-5 IMPLEMENTATIONS.

Model	**FPGA16[19]**	**FPGA17[17]**	**FPGA17[18]**	**FPGA18[20]**	**Proposed**
Design method	Sequential (binary)	Sequential (binary)	Sequential (binary)	Sequential (binary)	Parallel (stochastic)
FPGA platform	Zynq XC7Z020	Virtex7 VX690T	Virtex7 485t	Zynq zc706	Arria10 GX1150
Frequency(MHz)	100	100	–	166	150
Latency(us)	7916	94.2	960	1600	3.4
Kilo-Inferences per second (KIPS)	0.13	10.6	1.04	0.625	294.1
Performance (KIPS/MHz)	0.001	0.1	–	0.003	1.96
Power(W)	–	25.2	0.47	10.98	21
Energy efficiency(KI/J)	–	0.42	2.21	0.056	14
Logic used (LUT/ALM)	9682	233K	7204	39837	343.4K
Number of DSP used	2.4	2907	574	59	0
Memory Blocks used	6.16	477	343.3	97	0
Area efficiency (Inferences / MHz / ALM)	–	–	–	–	0.006

South Pacific Design Automation Conference (ASP-DAC). IEEE, 2017, pp. 696–701.

[14] A. Alaghi and J. P. Hayes, “Exploiting correlation in stochastic circuit design,” in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 39–46.

[15] A. Morro, V. Canals, A. Oliver, M. L. Alomar, F. Galán-Prado, P. J. Ballester, and J. L. Rosselló, “A stochastic spiking neural network for virtual screening,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 4, pp. 1371–1375, April 2018.

[16] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, “Throughput-optimized fpga accelerator for deep convolutional neural networks,” *TRETS*, vol. 10, pp. 17:1–17:23, 2017.

[18] Z. Li, L. Wang, S. Guo, Y. Deng, Q. Dou, H. Zhou, and W. Lu, “Laius: An 8-bit fixed-point cnn hardware inference engine,” in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, Dec 2017, pp. 143–150.

[19] S. I. Venieris and C. Bouganis, “fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs,” in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 40–47.

[20] S.-S. Park, K.-B. Park, and K. Chung, “Implementation of a CNN accelerator on an embedded soc platform using sdsoc,” 02 2018, pp. 161–165.