# Design and Implementation of a Highly Accurate Stochastic Spiking Neural Network

Chengcheng Tang
*Dept. of Electrical and Computer Engineering*
*University of Alberta*
Edmonton, Canada
ctang8@ualberta.ca

Jie Han
*Dept. of Electrical and Computer Engineering*
*University of Alberta*
Edmonton, Canada
jhan8@ualberta.ca

***Abstract*—The emergence of spiking neural networks (SNNs) provide a promising approach to the energy efficient design of artificial neural networks (ANNs). The rate encoded computation in SNNs utilizes the number of spikes in a time window to encode the intensity of a signal, in a similar way to the information encoding in stochastic computing. Inspired by this similarity, this paper presents a hardware design of stochastic SNNs that attains a high accuracy. A design framework is elaborated for the input, hidden and output layers. This design takes advantage of a priority encoder to convert the spikes between layers of neurons into index-based signals and uses the cumulative distribution function of the signals for spike train generation. Thus, it mitigates the problem of a relatively low information density and reduces the usage of hardware resources in SNNs. This design is implemented in field programmable gate arrays (FPGAs) and its performance is evaluated on the MNIST image recognition dataset. Hardware costs are evaluated for different sizes of hidden layers in the stochastic SNNs and the recognition accuracy is obtained using different lengths of stochastic sequences. The results show that this stochastic SNN framework achieves a higher accuracy compared to other SNN designs and a comparable accuracy as their ANN counterparts. Hence, the proposed SNN design can be an effective alternative to achieving high accuracy in hardware constrained applications.**

***Index Terms*—Field programmable gate arrays (FPGAs), priority encoder, stochastic computing, spiking neural networks.**

## I. Introduction

Spiking neural networks (SNNs) are a type of artificial neural networks (ANNs) that mimic the functions of the brain. [1]. An SNN encodes information in spike trains (that is, a series of logic “0” and “1” signals) that propagate through inter-connecting synapses between layers of neurons. A spike is generated only when the neuron’s membrane potential exceeds a certain threshold. Other than that moment, the synapses remain inactive and consume little energy.

There have been various attempts to implement SNNs on different types of hardware platforms [2], including application-specific integrated circuits (ASICs) and general processors, that led to some successful SNN-based designs such as the TrueNorth [3], Loihi [4], and SpiNNaker [5]. Because of the inherent massively-parallel structure, field programmable gate arrays (FPGAs) have also been considered for SNN implementations as a configurable and relatively cost-effective device. However, due to the high demand of hardware resources, special schemes such as an event-driven architecture have to be used to accommodate reasonably-sized SNNs into the FPGAs. Such schemes inevitably compromise the performance that an SNN can potentially achieve. For example, the FPGA-based design of SNNs in [6] achieves an accuracy of 92% on the MNIST handwritten digit classification and an accuracy of 97.06% at a rate of 161 frames per second is obtained in [7], which lag behind the performance of ANNs in accuracy.

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada (Project Number: RES0048688).

When looking into ways to reduce the hardware cost of neural networks, stochastic computing (SC) becomes appealing due to the simple logic used for complex computation. Unlike the conventional binary encoding, SC operates on randomly generated binary sequences, in which the probability of “1” is used to encode a number [8]. In SC, some complex arithmetic operations can be realized by a series of simple bitwise operations. The behaviour of SNNs exhibits many interesting analogues to SC [9]. Both methods convert complex tasks into simple operations over a certain period of time. As such, the outcome of one bit operation has no significance, but only the statistical outcome within a time interval matters. Because of the similar working mechanism of SC and rate-encoded SNNs, designs for SC, such as those for stochastic sequence generation, can also be applicable to SNNs and thus provide opportunities for improving the performance of stochastic SNNs [10]. In SNNs, the information carrier, spike trains, can be interpreted as stochastic sequences in SC.

However, stochastic SNNs suffer from two main problems: a relatively low information density and a high demand for hardware resources. The former is due to the fact that a long sequence length has to be used for encoding numbers to achieve adequate accuracy. The latter is due to the large number of neurons and the interconnections to be instantiated on hardware. This is especially the case for fully-connected neural networks (FCNNs), for which the number of connections grows quadratically with the number of neurons, i.e., $O(N^2)$, as there are $N^2$ connections between two layers with $N$ neurons in each layer.

This work attempts to mitigate these two problems by introducing a design using priority encoders (PEs). PEs are widely used to convert the input bit with the highest priority into its corresponding index in the binary format. Such a mechanism reduces the connections between neurons and uses the index to retrieve the right data in memories, which leads to substantial hardware savings. Considering the fact that a PE only transmits the highest priority bit to the output, a first-in, first-out (FIFO) block and priority resolving circuit (PRI) [11] are introduced to cope with the potential priority conflict.

The main contributions of this work include: (1) A design framework is outlined for stochastic SNNs that utilizes one random number generator (RNG) in the input layer and reduces the connections between layers from $O(N)$ to $O(\log N)$ by using a PE. (2) To resolve the problem that several spikes can possibly arrive at the same clock cycle in a hidden layer, a FIFO and PRI are introduced so that no spike will be discarded in the computation. (3) This framework is implemented on FPGAs; the hardware cost and accuracy are evaluated to show its efficacy in various aspects.

## II. PRELIMINARIES

Stochastic SNNs simulate the brain's function. A typical neuron in the brain takes information from the preceding neurons through its dendrites, processes it and then exports the processed information to the succeeding neurons through its axon. The message carriers are actually a series of electrical impulses that travel between different neurons. SNNs mimic this mechanism by operating on these impulses, called spike trains. A neuron in SNNs uses several inputs to receive spike trains from other neurons. Each of these connections is associated with a weight. The neuron updates its internal state and generates its own spike trains based on a predefined function of these input spikes and weights. During this process, the membrane voltage in a neuron will be reset to an initial value (usually zero) once the integration exceeds a predetermined threshold. In the meantime, a spike is generated to trigger the integration process in succeeding neurons.

### *A. Neuron Models in SNNs*

There are various types of neurons in SNNs, such as the Izhikevich's model [12], the FitzHugh-Nagumo model [13], the leaky integrate & fire (LIF) model. Most of them are developed by understanding the biochemical activities of the $\text{Na}^+$ and $\text{K}^+$ ions in the brain, which are generally expressed as differential equations, thus making them computationally intensive [14]. For hardware design, a simple model is beneficial if it preserves the same functionality. Thus, the commonly used integrate & fire (IF) model is considered in this work. It can be shown that this model in SNNs is statistically equivalent to the ReLU model in conventional ANNs [15]. It was also shown in [16] that through proper weight and threshold balancing, a ReLU based ANN can be converted into an SNN with nearly no accuracy loss.

The IF model for a given neuron $i$ in SNNs can be expressed as [2]

$$\frac{dx_i(t)}{dt} = \sum_j W_{i,j} S_j(t), \tag{1}$$

where $x_i(t)$ is the membrane voltage of this neuron, $W_{i,j}$ is the weight of the $j$th input connection and $S_j(t)$ is a function of $t$ for the spike train on this connection. When there is a spike at any given time $t = t_k$, then $S_j(t_k) = 1$; otherwise $S_j(t_k) = 0$. The output spike train of this neuron is created based on the value of $x_i(t)$. When $x_i(t)$ reaches a threshold $X_{\text{th}}$, a spike is generated and $x_i(t)$ is reset to $V_{\text{rst}}$; otherwise, it continues evolving based on (1) and no spike is generated.

Obviously, a spike itself has no significant meaning, but the interval between spikes conveys useful information. For the rate coding mechanism in SNNs, a more densely distributed spike train means that the membrane voltage of a neuron reaches the threshold value more often and thus stands for a "strong" signal intensity. Relatively, a sparsely distributed spike train indicates a "weak" signal intensity. Therefore, the number of spikes transmitted during a given period of time (denoted as $X$) quantitatively indicates how strong a signal is. In statistics, $X$ is proportional to the expectation of the total variation of signal $x$ for a given period of time. This relationship is given by

$$E(\Delta x_i) = E\left(\int \sum_j W_{i,j} S_j(t)\right) = \sum_j W_{i,j} \cdot X_j, \tag{2}$$

where $\Delta x_i$ is the total variation of signal $x_i$ for a given period of time. Considering that $X_i$ is always non-negative and $X_{th}$ is a constant, it can be found from (2) that

$$X_i = E(\Delta x_i)/X_{th} \propto \max\left(0, \sum_j W_{i,j} X_j\right), \tag{3}$$

which is actually the ReLU function [16].

This analysis shows that the computation in SNNs realizes the same function as a ReLU based ANN. Hence, the weight parameters of SNNs can be inherited from a well trained ReLU based ANN. In the SNNs, a weighted sum is actually decomposed into an integration over a certain period of time, which is much easier to implement because multiplication is no longer needed.

Usually in an SNN application, only few neurons have relatively high activation values, whereas most of them are slightly invoked or even remain inactive. In other words, the average information density of these spike trains is very low. To mitigate this issue, we propose to employ priority encoders as the inter-connecting units between neurons in stochastic SNNs, as discussed in Section III.

### *B. Priority Encoder*

A priority encoder is a module that converts inputs into binary signals that indicate the position or index of the input bit with the highest priority that is "1". Fig. 1 illustrates a 4 to 2 priority encoder, by which the four input signals are encoded into 2-bit binary signals. Note that D0 has no impact on the output and it is reserved for the no-spike scenario in the inputs. A zero is also inserted at the beginning of the weights vector for each neuron so that when no spike is present, the PE outputs all "0"s and the neural potential does not change.

| D3 | D2 | D1 | D0 | Q1 | Q0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | x | 0 | 0 |
| 0 | 0 | 1 | x | 0 | 1 |
| 0 | 1 | x | x | 1 | 0 |
| 1 | x | x | x | 1 | 1 |

(a)

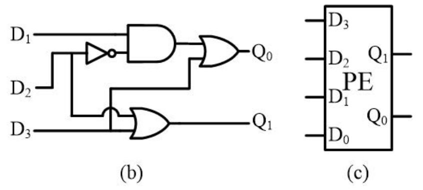

Fig. 1. A 4 to 2 priority encoder: (a) a truth table, (b) a logic structure, and (c) a symbol.

The output of a priority encoder is a number in the binary format indicating the position of the highest bit in input that is set to “1”. A PE with a large width can be constructed by using smaller units with a scaling method [11]. For example, a 64 to 6 PE can be constructed by four 16 to 4 PEs and one 4 to 2 PE. When connecting the spike trains to the inputs of a PE, the output signals can be used to find the index of the neuron from which a spike arrives. Meanwhile, it helps reduce the number of connections between the neurons in different layers from $O(N)$ to $O(\log N)$.

## III. A Design Framework for Stochastic SNNs

Fully-connected neural networks are widely used in many applications because of their versatility. However, it is often computationally expensive due to the huge number of connections between layers of neurons. Thus, it will serve as a reference structure in this paper for design validation.

### A. Design of the Input Layer

The input layer takes digital information obtained from the outside world (for example, the intensities of image pixels) and converts it into spike trains. As shown in Fig. 2(a), each spike train is generated by comparing the data from one RNG and one input channel, in a similar way to how a stochastic sequence is generated in SC. The resulting sequence follows a Poisson distribution as the simplest stochastic model of neuronal firing [17]. However, this model is not optimal for hardware implementation, because each input has to be equipped with one RNG. In addition, when the input datasets are sparse, the most part of the spike trains will remain zero. Therefore, no spike is generated for the most of time and thus no state transition is triggered.

One way to address this problem is to use the incremental accumulation of the input signal to map uniformly distributed random numbers to a cumulative distributed function (CDF) [18]. As shown in Fig. 2(b), the input data $x$ on an axis with the interval lengths corresponding to the data values are accumulated into $F$. For example, if $x_1 = 2$, $x_2 = 3$ and $x_3 = 5$, then $F_1 = 2$, $F_2 = 5$ and $F_3 = 10$. Then a random number would always fall into one of these intervals, namely $0 < F \leq 2$, $2 < F \leq 5$, $5 < F \leq 10$. The probability that the random number falls into an interval is proportional to the value in input data with the same index.

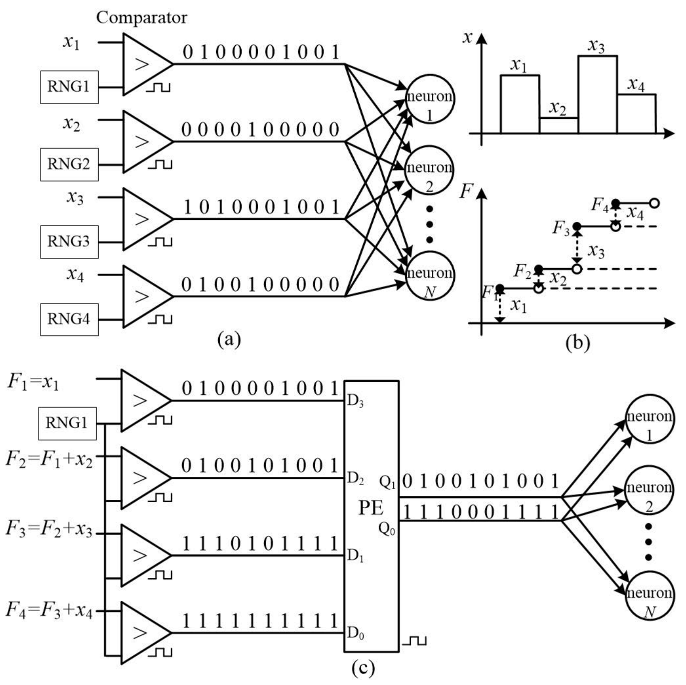

Fig. 2. The input layer of the stochastic SNNs. (a) Conventional design, (b) Illustration of the CDF ($F_i = \sum_{j=1}^{i} x_j$), (c) The proposed design.

Therefore, the input layer of the proposed SNN is implemented by comparing the CDF of input signals with a common RNG, as shown in Fig. 2(c). The resulting spike trains are forwarded to a PE and then the output is sent to the subsequent layer of neurons. In this way, the output binary signal Q from the PE represents the index of the section in the CDF a random number falls into. Since the random number is uniformly distributed between one and its maximum value, Q follows the same distribution with the input intensity, i.e.,

$$P(Q = \text{index}(x_i)) = \frac{x_i}{\sum_k x_k}. \tag{4}$$

This structure has three unique features. First, the RNG is shared by every input channel. Inevitably, the required bit-width of the shared RNG is larger than that of the single RNGs in traditional spike train generation. For example, if there are $q$-channels of inputs and each of them is $p$-bit wide, then we need a $p + \log(q)$ bit wide signal to avoid potential overflow. Nevertheless, hardware savings can still be achieved compared with the use of multiple smaller RNGs because the size only increases logarithmically with the number of input channels. Second, by properly assigning the range of the random number (i.e., by setting it to be less than $F_{\max}$), it will always be located within one section of the CDF, which means that spikes are generated in every clock cycle of the digital system. As shown in Fig. 3(c), the sequence generated by the comparator for F4 (leading to the input of the PE with the least priority) is a sequence of all “1”s. Hence, at least one spike is passed through the PE in every clock cycle. Lastly, by utilizing a PE, the number of connections to the neurons in the subsequent layer is reduced from $O(N)$ to $O(\log N)$ by converting the number of connections into the binary representation, resulting in significant hardware savings. Since the outputs of the PE can be used as address signals by the succeeding neurons to find the corresponding weight during computation, it is convenient to utilize distributed memory on FPGA to store weights along with the neuron.

## B. Design of the Hidden Layers

A hidden layer receives spike trains from the previous layer and propagates these spikes to the next layer according to some predefined propagation rules. Unlike the input layer, only one spike needs to be processed in every clock cycle. Once the spike train departs the input layer, the spikes on different channels might arrive at the same time. Therefore, a structure is designed for the hidden layers of stochastic SNNs to ensure that no spike is discarded, as shown in Fig. 3.

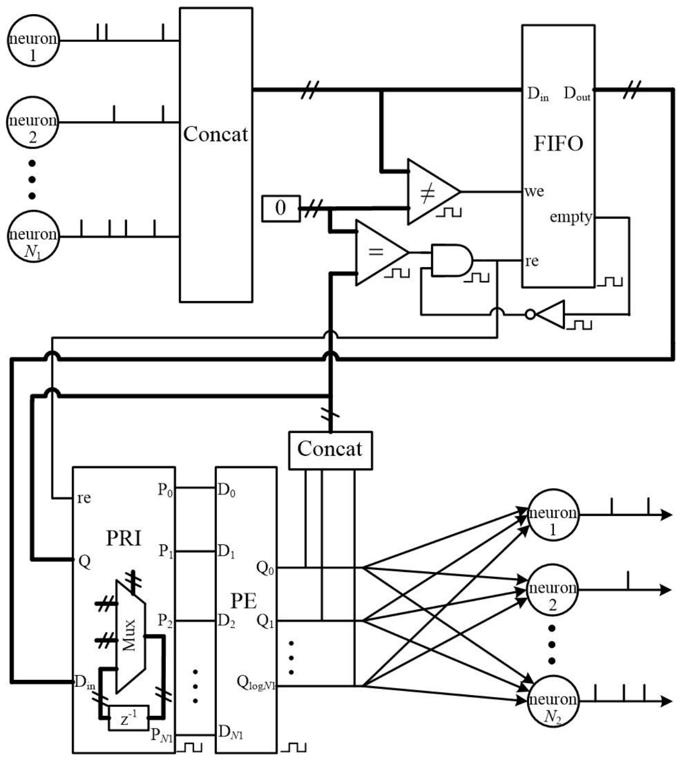

Fig. 3. A hidden layer of the stochastic SNNs.

The spike trains from the previous layer are first concatenated into a data bus and then sent to a FIFO. Whenever a spike appears on the data bus, the FIFO intakes the data on the bus and stores it in its queue. The output of the FIFO is connected to a PRI followed by a PE. Then the encoded results from the PE are sent back to the FIFO and PRI to release the next available data or clear the bit that has been encoded. Note that although two or more spikes can arrive at the same time, the spike trains are still sparse. The FIFO only accepts a signal that does not contain all zeros; when all the spikes in the data bus have been processed, a new read enable (re) is generated to release the next data. The PRI circuit is used to avoid repeated encoding of the same bit by collaborating with the PE. When the PE finishes encoding the bit with the highest priority, the PRI then clears this bit so that the PE can move on to encode the second highest priority bit. By using this process, all spikes in the train are completely processed and propagated to the neurons in the next layer.

## C. Design of the Output Layer

The function of the output layer is to produce computation results to an external system. Since stochastic SNNs operate on spike trains, which have no standard form of data representation, that is, multiple combinations of sequences can be used to encode the data. As a result, the output layer needs to convert the data back into the binary format. Usually, when the stochastic SNNs are used for some classification or identification applications, it is the comparison result of the output values that has a practical meaning and the neuron with the largest membrane voltage value needs to be identified. Hence, the output layer utilizes the structure shown in Fig. 5 for the data conversion.

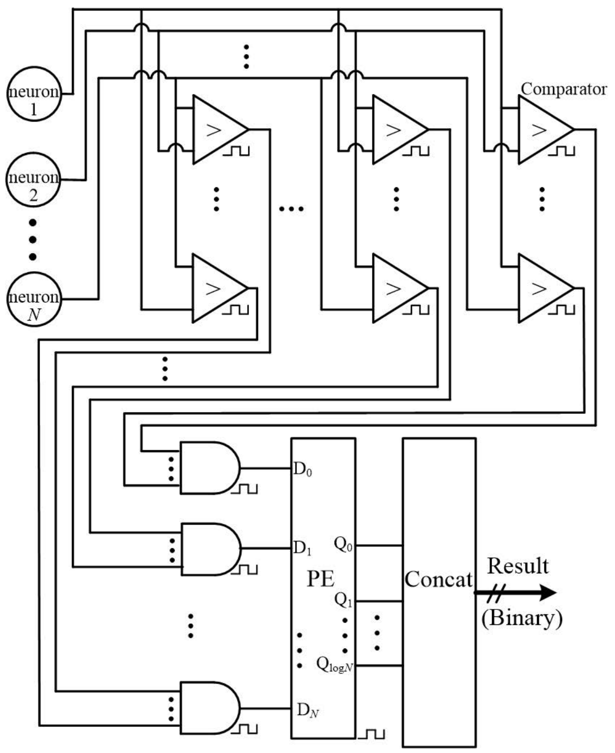

Fig. 4. The Output layer of the stochastic SNNs.

Unlike the neurons in previous layers, the neurons in the output layer integrate the incoming spike trains but no longer need to propagate them. The membrane voltage values of these neurons are sent to a comparison matrix, by which the neuron $i$ is compared to all the other neurons in column $i$. The $N-1$ comparison results are connected to an AND gate to indicate if neuron $i$ has the largest membrane voltage value. The results of $N$ AND gates are then connected to a PE to convert them into the binary format, which is the identification result and can directly be used for verification.

## IV. Hardware Implementation and Performance Evaluation

Hardware implementation of the stochastic SNNs is conducted on an FPGA to verify the efficacy of this design. The widely used MNIST dataset is selected for evaluation. The training of the dataset is performed using a ReLU based ANN in Matlab. The trained weight values are then exported to the distributed block RAM on the FPGA. However, the ANNs in Matlab use floating-point representation while the accumulators in stochastic SNNs work on a fixed-point format. To reduce accuracy loss, the weight values after training are scaled up by a fixed number and only the integer parts are stored in memory. To accurately simulate the computation in ANNs, the threshold $X_{th}$ in (3) is selected to be 1.0 for the neuron membrane voltage reset in stochastic SNNs. The number of spikes generated by a neuron during an inference period is the same as the integer part of the computation result by ANNs.

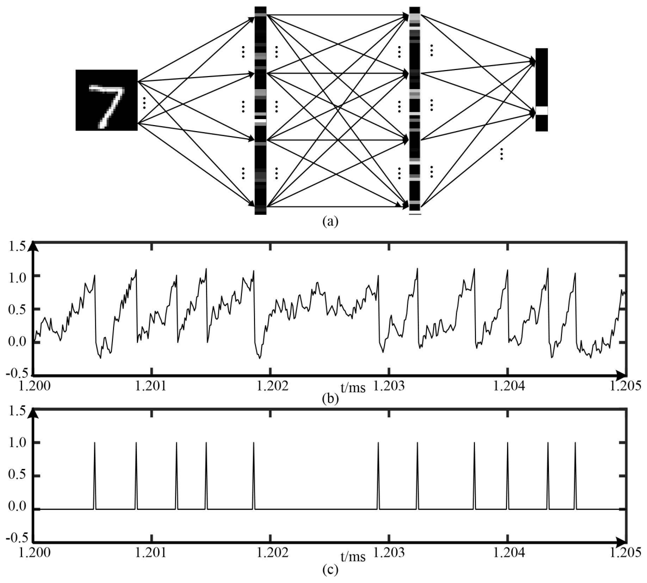

Fig. 5. Simulation process for the recognition of an image of digit 7. (a) The neuron/pixel intensity as shown in the grey stripes. (b) The membrane voltage value for a neuron in a hidden layer. (c) The corresponding spike train for the same neuron.

Three fully-connected stochastic SNNs with different sizes of hidden layers, i.e. 784-63-63-10, 784-127-127-10 and 784-255-255-10, are implemented on FPGAs. Fig. 5 illustrates the simulation results of the 784-63-63-10 network for the recognition of an image for the number 7. The grey value represents the intensity of a neuron/pixel (with a brighter pixel/neuron standing for a stronger signal intensity). Fig. 5 (b) and (c) showcase the membrane voltage variations and how the corresponding spike train is generated for a neuron in the hidden layer.

It can be seen that the number in this image has been correctly identified, i.e., the 7th neuron in the output layer has the largest brightness.

To comprehensively assess the performance of hardware cost and recognition accuracy, these three SNNs are evaluated on the 10,000 images in the test set of the MNIST. The FPGA for hardware implementation and experimentation is the Xilinx Virtex7 xc7vx485t. The system works at a clock frequency of 100 MHz and 10,000 random numbers are used for each image recognition. In other words, each image recognition takes only 100 $\mu$s and a total of one second is needed for completing the whole MNIST test set. Hardware utilization and recognition results are summarized in Table I, in which the target accuracy is obtained from the ANNs and the percentage under each hardware cost is the ratio of used and available resources in the FPGA.

TABLE I
Hardware utilization and recognition accuracy of three stochastic SNNs

| **Structure Size** | LUT | FF | BRAMs | Recognition Accuracy | Target Accuracy |
|---|---|---|---|---|---|
| **784-63-63-10** | 16555 (5.45%) | 20347 (3.36%) | 101.5 (9.85%) | 97.37% | 97.39% |
| **784-127-127-10** | 24089 (7.93%) | 24982 (4.11%) | 199.5 (19.37%) | 97.88% | 97.89% |
| **784-255-255-10** | 39015 (12.85%) | 34217 (5.64%) | 395.5 (38.4%) | 98.24% | 98.23% |

LUT: look-up-table; FF: flip-flop;
BRAMs: block-random-access-memories (18k bits each)

It can be seen that in all these three cases the recognition accuracy of stochastic SNNs is very close to that obtained from the ANNs. It can even be slightly higher than the reference accuracy, which indicates that the stochastic SNNs almost reproduce the computation results in ANNs, as shown analytically in Section II. On the other hand, it is not surprising to see that the hardware cost of stochastic SNNs increases with the number of neurons in the hidden layers, especially in the number of block-random-access-memories (BRAMs), as shown in Table I. The reason is that BRAMs grow quadratically in $O(N^2)$ while the number of connections between layers increases in $O(N\log N)$ with the number of neurons, $N$, in this design.

To better assess the performance in different aspects, the comparison between this design and some others is illustrated in Table II. The results show that this design achieves a higher accuracy with a fast recognition rate in terms of processing time per each processed image. Its hardware cost, however, is not advantageous over the other designs. This is due to the massively parallel architecture and the use of 32-bit long data format for the weight values in the implementation. Thus, there is room for improvement in future work.

The size of the hidden layer is not the only factor that determines the recognition accuracy. The duration of time for each image identification and the length of stochastic sequences also have a significant impact on the accuracy. To evaluate their effects, experiments using different stochastic

TABLE II
HARDWARE UTILIZATION AND RECOGNITION ACCURACY COMPARED TO THREE OTHER SNN DESIGNS

| | Han et al., [7] | Gupta et al., [19] | Liang et al., [20] | This Design |
|---|---|---|---|---|
| **Platform** | Xilinx ZC706 | Xilinx XC6VLX240T | Xilinx Virtex7 VC7VX485T | Xilinx Virtex7 XC7VX485T |
| **Structure** | 784×1024 ×1024×10 | 784 × 16 | 784 × 512 × 10 | 784×255 ×255×10 |
| **Weight Precision** | 16-bit fixed-point | 24-bit fixed-point | 8-bit fixed-point | 32-bit fixed-point |
| **LUT** | 5381 | 56230 | 16324 | 39015 |
| **FF** | 7309 | 23238 | 11612 | 34217 |
| **BRAMs** | 40.5+ external DDR | 16 | - | 395.5 |
| **Accuracy** | 97.06% | - | 96% | 98.24% |
| **Time/Image** | 6210 $\mu$s | 500 $\mu$s | 2.8 $\mu$s | 100 $\mu$s |

sequence lengths for each image identification on the 784-255-255-10 network are performed. The results are shown in Table III.

Note that the duration of time along with the clock frequency of FPGAs determines the length of stochastic sequences. For example, a duration of 10 $\mu$s means 1000 clock cycles for a frequency of 100 MHz, which is also the sequence length used in the experiment. A longer duration of the identification time leads to a higher accuracy but the difference is not very significant. In this example, a ten times longer sequence length increases the accuracy by only 1.58% and it tends to saturate at the accuracy obtained by the ANN counterpart. In cases with a lesser accuracy requirement, a shorter identification time for each image can be considered for faster processing.

TABLE III
RECOGNITION ACCURACY USING DIFFERENT DURATIONS OF TIME FOR IMAGE IDENTIFICATION

| **Duration of time (Sequence length)** | 10 $\mu$s (1k bits) | 20 $\mu$s (2k bits) | 50 $\mu$s (5k bits) | 100 $\mu$s (10k bits) |
|---|---|---|---|---|
| **784-255-255-10** | 96.66% | 97.57% | 98.01% | 98.24% |

## V. CONCLUSION

This paper presents a hardware design framework for SNNs inspired by the notion of stochastic computing. This framework utilizes the cumulative distribution function of the input signal for spike train generation and a priority encoder to convert these spike trains into index-based signals. Such configuration reduces the connections between layers of neurons from $O(N^2)$ to $O(N\log N)$ for efficient hardware design. This stochastic SNN is implemented on FPGAs for classifying the MNIST dataset. The experimental results show that it achieves nearly the same accuracy as its ANN counterparts at a high processing rate. Hardware utilization of the stochastic SNNs with different sizes of hidden layers and the impact of various stochastic sequence lengths on accuracy are investigated, which raises challenges and opportunities for further improving the design framework in future work.

## REFERENCES

[1] S. R. Kulkarni, A. V. Babu and B. Rajendran, "Spiking neural networks - algorithms, hardware implementations and applications," in *Proc. 2017 IEEE 60th Int. Midwest Symp. on Circuits Syst. (MWSCAS)*, Boston, MA, USA, 2017, pp. 426-431.

[2] M. Bouvier *et al*, "Spiking neural networks hardware implementations and challenges: a survey," *ACM Journal on Emerging Technologies in Computing Systems,* vol. 15, no. 2, pp. 1-35, June 2019.

[3] P. A. Merolla *et al*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science,* vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[4] M. Davies *et al*, "Loihi: a neuromorphic manycore processor with on-chip learning," *IEEE Micro,* vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[5] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *Proc. 2008 IEEE Int. Joint Conf. on Neural Networks,* Hong Kong, China, 2008, pp. 2849–2856.

[6] D. Neil, and S. Liu, "Minitaur, an event-driven FPGA-based spiking network accelerator," *IEEE Trans. Very Large Scale Integr. Syst.,* vol. 22, no. 12, pp. 2621–2628, Dec. 2014.

[7] J. Han, Z. Li, W. Zheng, and Y. Zhang, "Hardware implementation of spiking neural networks on FPGA," *Tsinghua Sci. Technol.,* vol. 25, no. 4, pp. 479–486, Aug. 2020.

[8] Y. Liu, S. Liu, Y. Wang, F. Lombardi and J. Han, "A survey of stochastic computing neural networks for machine learning applications," in *IEEE Trans. on Neural Networks and Learning Systems,* vol. 32, no. 7, pp. 2809-2824, July 2021.

[9] S. C. Smithson, K. Boga, A. Ardakani, B. H. Meyer, and W. J. Gross, "Stochastic computing can improve upon digital spiking neural networks," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS),* Dallas, TX, USA, Oct. 2016, pp. 1–6.

[10] S. Liu, W. J. Gross and J. Han, "Introduction to dynamic stochastic computing," *IEEE Circuits Syst. Mag.,* vol. 20, no. 3, pp. 19-33, 2020.

[11] X. Nguyen, H. Nguyen and C. Pham, "A scalable high-performance priority encoder using 1D-array to 2D-array conversion," *IEEE Trans. Circuits Syst. II, Exp. Briefs,* vol. 64, no. 9, pp. 1102–1106, Sep. 2017.

[12] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.,* vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[13] R. FitzHugh, "Impulses and physiological states in models of nerve membrane," *Biophysics Journal,* vol. 1, pp. 445-466, 1961.

[14] R. Borwankar, A. Desai, M. R. Haider, R. Ludwig and Y. Massoud, "An analog implementation of FitzHugh-Nagumo neuron model for spiking neural networks," in *Proc. 2018 16th IEEE Int. New Circuits and Syst. Conf. (NEWCAS),* Montreal, QC, Canada, 2018, pp. 134-138.

[15] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran and K. Roy, "A comprehensive analysis on adversarial robustness of spiking neural networks," in *Proc. 2019 Int. Joint Conf. on Neural Networks (IJCNN),* Budapest, Hungary, 2019, pp. 1-8.

[16] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proc. 2015 Int. Joint Conf. on Neural Networks (IJCNN),* Killarney, Ireland, 2015, pp. 1-8.

[17] M. Fatahi, M. Shahsavari, M. Ahmadi, A. Ahmadi, and P. Devienne, "evt_MNIST: A spike based version of traditional MNIST an event-based MNIST," in *Proc. Int. Conf. on New Research Achievements in Electrical and Computer Engineering,* Teheran, Iran, 2016.

[18] M. Alawad, H. Yoon and G. Tourassi, "Energy efficient stochastic-based deep spiking neural networks for sparse datasets," in *Proc. 2017 IEEE Int. Conf. on Big Data (Big Data),* Boston, USA, 2017, pp. 311-318.

[19] S. Gupta, A. Vyas and G. Trivedi, "FPGA implementation of simplified spiking neural network," in *Proc. 2020 27th IEEE Int. Conf. on Electron., Circuits and Sys. (ICECS),* Glasgow, UK, 2020, pp. 1-4.

[20] M. Liang, J. Zhang and H. Chen, "A 1.13$\mu$J/classification spiking neural network accelerator with a single-spike neuron model and sparse weights," in *Proc. 2021 IEEE Int. Symp. on Circuits and Sys. (ISCAS),* Daegu, South Korea, 2021, pp. 1-5.