A Stochastic Computing Sigma-Delta Adder Architecture for Efficient Neural Network Design

Nikos Temenos, *Member, IEEE*, and Paul P. Sotiriadis, *Fellow, IEEE*

***Abstract*— A stochastic computing (SC) adder architecture, based on sigma-delta modulation is introduced. The operation principle of the stochastic computing sigma-delta (SCSD) adder is presented and the adder is modeled using Markov Chains resulting in the derivation of output's first-moment statistics and the demonstration of its fast-convergence. The SCSD's single-bit output enables the connection to existing Stochastic Finite-State Machines realizing non-linear functions and the formation of cascade processing structures appropriate for efficient realizations of SC artificial neurons. To demonstrate the proposed adder's efficacy, a SCSD adder-based neuron was designed and used as a building block of a SCSD Multi-Layer Perceptron (MLP). The computational accuracy of the SCSD MLP and the hardware resources it requires are compared to those of the Fixed-Point arithmetic implementation, highlighting the effectiveness and significant area savings of the proposed SC approach. Furthermore, the accuracy and hardware requirements of the proposed SCSD MLP are also compared to those of corresponding SC architectures in the literature.**

***Index Terms*— Stochastic computing sigma-delta adder, stochastic computing neural network, stochastic computing multi-layer perceptron.**

I. Introduction

THE prominent field of Artificial Intelligence (AI) has made the presence of Machine Learning (ML) algorithms and Neural Network (NN) architectures important in modern digital signal processing cores (DSP) [1], [2], [3]. Conventional computing methods used for the hardware realizations of AI-related tasks, struggle to simultaneously fulfill size, low-power consumption, high-computational performance and massive parallelism constraints [3], [4]. To overcome such design limitations, unconventional computing paradigms are employed with Stochastic Computing (SC) being an effective approach [1], [3], [4], [5].

Deviating from the binary arithmetic's processing, SC operates on sequences of logic 1s and 0s [6]. The bit-serial processing enables the realization of arithmetic operations and highly-complex functions using logic gates and standard cells. Further, the SC's probabilistic nature makes it robust to bit-flips and soft errors [4]. These advantages of SC have been exploited in hardware realizations of ML algorithms and NN architectures given their massive parallelism needs along with their small error tolerance [1], [3], [7], [8].

Within the SC-based DSP cores, the multiply-and-add operation is the most important one. Each multiplication between two sequences is realized using an AND or an XNOR gate according to the SC number representation used [6]. The addition part is typically realized using two-input adder-tree structures or multi-input adders. Typical two-input SC adders scale the addition's result by a factor of two, reducing also the output sequence's resolution by the same factor [9]. This forces each subsequent layer within the adder-tree to increase the addition's scaling in increasing powers of two. To compensate for the resolution drop, the sequence length should be increased in powers of two according to the number of layers within the adder-tree, resulting in increased latency and total energy consumption [10]. Moreover, the adder-tree's scaled output cannot be applied directly to non-linear functions as it has to be properly re-scaled fist, requiring additional resources [11].

To address the adder-tree's scaling challenges, multi-input adders were considered for use in SC, with the accumulative parallel counter (APC) [12] being the most popular one [1], [13], [14], [15]. The APC accumulates deterministically all input sequences in parallel, producing the result in binary format. In SC-based cascaded computations however, the APC's binary output introduces the following design challenges; 1) It limits the applicability of existing single-bit input/output Stochastic Finite-State Machines (SFSMs) realizing highly-complex functions including non-linear ones [16], [17] and 2) It requires the conversion of the APC's binary output to a stochastic sequence if other SC arithmetic operations, e.g. multiplications, follow [18]. Another promising approach addressing the adder-tree's scaling challenges was proposed in [10] and was successfully applied in the realization of cascaded multiply-and-add operations [5], [10]. It offers improved accuracy with short input sequence lengths, at the cost of additional registry elements per adder used, which may potentially impact the hardware resources when designing large adder-trees.

Both adder-tree and multi-input adder design strategies have been explored within the context of SC for the realization of Multi-Layer Perceptrons (MLP), which is a class of NNs, [15], [19], [20]. In [15], each neuron forming the MLP is realized using an APC followed by a multi-bit input single-bit output FSM approximating the *tanh* (BTanh) non-linear activation function, implemented as a binary up/down counter. However, in the BTanh's design the FSM's number of states affecting the *tanh*'s approximation is derived using numerical

Manuscript received 21 October 2022; revised 23 December 2022 and 6 February 2023; accepted 7 February 2023. Date of publication 10 February 2023; date of current version 20 March 2023. This article was recommended by Guest Editor J. Han. *(Corresponding author: Nikos Temenos.)*

The authors are with the Department of Electrical and Computer Engineering, National Technical University of Athens, 15780 Athens, Greece (e-mail: ntemenos@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2023.3243950.

Digital Object Identifier 10.1109/JETCAS.2023.3243950

experiments. In [19], a hybrid SC MLP is realized using an adder-tree structure composed of extended stochastic logic (ESL) adders [21] in the input layer and APCs in the remaining layers. This hybrid format encoding enables the on-line update of weights. On the other hand, the ESL adders require an additional binary-to-stochastic number converter for multiplexer's select signal, taxing the hardware resources [10], [21], while the adder-tree requires a triple modular redundant (TMR) binary search divider, resulting in large sequence lengths for its computation and stabilization phases [1], [19]. With respect to the activation functions, the same approach as in [15] was followed, but, a multi-input single-bit output FSM realizing the rectifier linear unit (ReLU) was used. Similar to [19], in [20] a gradient-based updating scheme was applied to a MLP, showing the effectiveness of the gradients' and the weights' on-line learning. Yet, in [19] no emphasis in the multiply-and-add operations is given.

Motivated by the limitations of existing SC adder design strategies, this work introduces a SC adder architecture that utilizes a first-order sigma-delta modulator (SDM). The proposed Stochastic Computing Sigma-Delta (SCSD) adder sums the weighted input sequences into a single-data bus and then employs an internal range conversion scheme. It offers the following advantages: 1) it operates on independent inputs, 2) the addition is done deterministically without additional random sources, 3) it is fast converging with small sequence lengths, 4) it enables cascaded operations to be realized efficiently with existing SC arithmetic circuits and 5) it allows the use of *any* single-bit input/output SFSM, thereby expanding the SC-based NN design space.

The proposed SCSD adder's operation principle is verified with mathematical analysis, supported by Markov Chain (MC) modeling, resulting in a deeper understanding of its stochastic dynamics and in the derivation of its first order statistics. To demonstrate its efficacy, the proposed SCSD adder is used for the realization of a SC artificial neuron, setting the foundation for the formation of a SC-based MLP.

The remainder of the proposed work is organized as follows. In Section II the essential notation of the stochastic numbers is provided. In Section III the proposed SCSD adder is analyzed in detailed and modeled as a MC. Section IV introduces the SC-based artificial neuron realized using the proposed SCSD adder, as well as a MLP formed of SCSD adder-based neurons. In Section V experimental results for two MLP architectures are shown in terms of computational accuracy, while the hardware efficiency is compared to that of the Fixed-Point (FxP) arithmetic. Moreover, the computational accuracy and the hardware efficiency of other SC MLP approaches compared to the proposed one are discussed. Finally, Section VI concludes the present work.

II. Stochastic Number Representation

A b-bit length binary number B is converted into a stochastic number (SN), i.e. a sequence of logic 1s and 0s, using a stochastic number generator (SNG). The SNG utilizes a random number source of b-bit length and on every clock cycle compares its value to the binary number B, for a total of $N = 2^b$ clock cycles, corresponding to the sequence length. The generated sequence $\{X_n\}_{n=1}^{N}$ is assumed to be formed of independent and identically distributed (i.i.d.) random variables (r.v.), where $n = 1, \ldots, N$ is the time index. The SN has probability $p_X \triangleq P_r(X_n = 1) = B/2^b$ and time-average value defined as

$$\widetilde{X}_N = \frac{1}{N}\sum_{n=1}^{N} X_n. \tag{1}$$

The b-bit length binary number B can be used to represent a negative-signed number. To avoid confusion, we use the notation $\hat{X} = p_X$ to represent the value of B and consequently that of the SN when it belongs in [0, 1], known as unipolar format. On the other hand, when the value of B is a negative-signed number, we use the following equation

$$\hat{X} = 2p_X - 1. \tag{2}$$

to extend the SN's range to $[-1, 1]$, known as bipolar format. To convert a SN back into its binary form, an accumulator of b-bits is used [6].

III. Stochastic Computing Sigma-Delta Adder

This section introduces the architecture of the proposed SCSD adder along with the corresponding analysis of the operation principle, the modeling using MCs, the error characteristics and a comparison of the hardware resources to those of existing approaches.

A. Architecture

The proposed multi-input single-bit output SCSD adder architecture is shown in Fig. 1. The input sequences $\{X_n^j\}_{n=1}^{N}$, $\{W_n^j\}_{n=1}^{N}$ with $j = 1, \ldots, k$ are assumed to be i.i.d. and independent to each other, while $\{Z_n\}_{n=1}^{N}$ is the output sequence. The XNOR gates are used to multiply the input sequences pairwise in bipolar format [6] resulting in k intermediate sequences $\{U_n^j\}_{n=1}^{N}$. Sequences $\{U_n^j\}_{n=1}^{N}$ are added deterministically using conventional binary arithmetic as shown in Fig. 1, without additional randomizing sources. This implies that the stochastic precision of sequence $\{Y_n\}_{n=1}^{N}$ is exclusively determined by the length, N, of the input sequences.

Assuming sequences $\{X_n^j\}_{n=1}^{N}$, $\{W_n^j\}_{n=1}^{N}$ carry signed number information, we do the conversion for all of them just once. Since Y_n is the weighted sum of k inputs, the range of Y_n is extended from $\{0, \ldots, k\}$ to $\{-k, \ldots, k\}$ using the transformation $V_n = 2Y_n - k$. Note that the multiplication operation is realized using only a left shift operation.

The bit-width c of Y_n is determined according to the number of inputs, k, and should be such that it can capture all incoming bits, namely $c = \lfloor \log_2 k \rfloor + 1$, where $\lfloor \cdot \rfloor$ is the floor function. On the other hand, the bit-width after the range conversion should be $c' = c + 1$, accounting for the signed value of V_n.

The first-order digital SDM (DSDM) within the SCSD adder architecture of Fig. 1, consists of an adder and an m-bit register with $m \geq c' + 1$ so as to capture the accumulation process of the first-order DSDM, followed by a most significant bit (MSB) selection block. The MSB block, replaces the quantizer in the system level model of a typical first order SDM shown in Fig. 2, which is a simplification of the comparison between

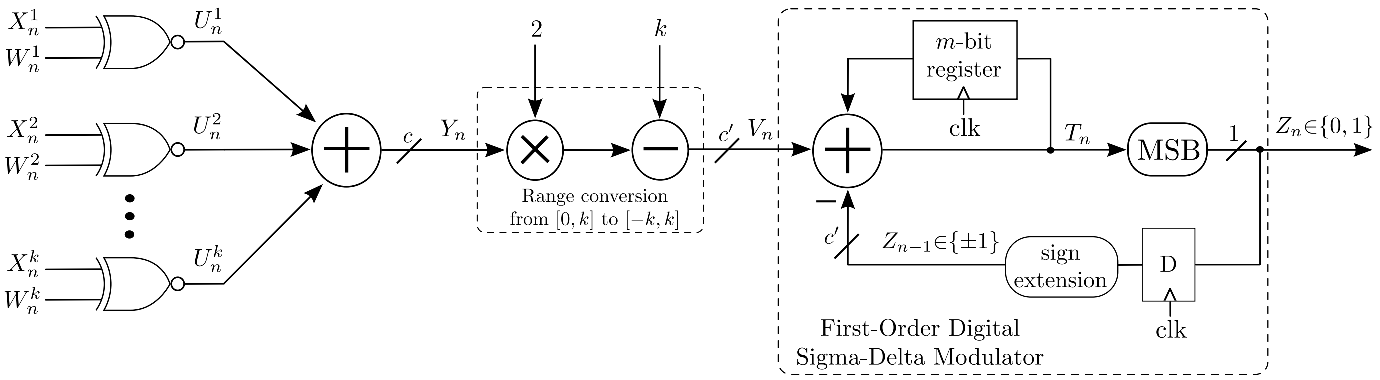

Fig. 1. Architecture of the proposed Stochastic Computing Sigma-Delta (SCSD) adder. The XNOR gates multiply pairwise the input sequences $\{X_n^j\}_{n=1}^N, \{W_n^j\}_{n=1}^N$ in bipolar format. The multiplication results are added to a single bus with the range of its represented value converted from $[0,k]$ to $[-k,k]$. The first-order digital SDM converts a high-bit-resolution signal into a single-bit one whose time-average approximates the sum-of-products. The architecture can be further simplified by pushing the range conversion into the SDM.

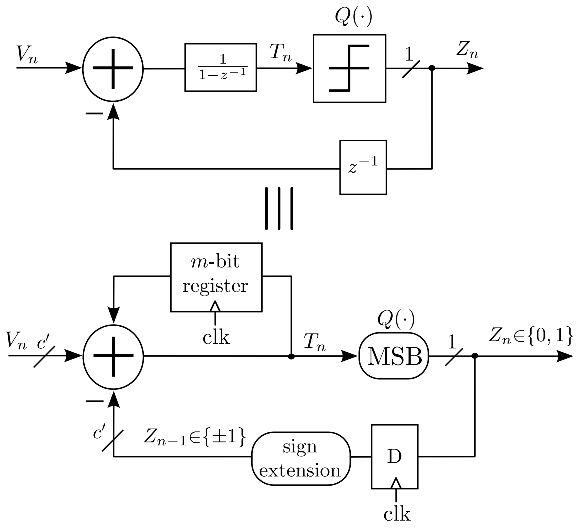

Fig. 2. Top: system level model of a first-order Sigma-Delta Modulator. Bottom: realization of the first-order Digital Sigma-Delta Modulator. The quantizer block, is replaced by the selection of the most significant bit.

the register's current value and zero [22], [23]. Therefore, considering the signed representation of T_n, the MSB's operation implements a function $Q(\cdot)$ as

$$Q(T_n) = \begin{cases} 1, & \mathrm{MSB}(T_n) = 1 \\ 0, & \mathrm{MSB}(T_n) = 0 \end{cases}. \tag{3}$$

Note that in the DSDM shown in Fig. 1, the sign extension operation feeds back $Z_{n-1} = 1$ when $Q(T_n) = 1$ and $Z_{n-1} = -1$ when $Q(T_n) = 0$.

The register's initial value T_0 can be any one within $\mathcal{T} = \{0, 1, \ldots, M-1\}$, where $M = 2^m$ is the number of states, and the DSDM's current state T_n is updated as

$$T_n = \max\left\{0, \min\left\{T_{n-1} + V_n - Z_{n-1}, M-1\right\}\right\}. \tag{4}$$

The $\max(\cdot)$ and $\min(\cdot)$ functions are used here to denote the register's natural saturation to states 0 and $M-1$, given that they cannot be exceeded. The SDM's output and consequently that of the SDSC adder is

$$Z_n = Q(T_n). \tag{5}$$

If the register of the DSDM does not saturate, the time-average of the output $\mathbb{E}\left[Z_n^j\right] \approx \widetilde{Z}_N^j$ equals (or approximates in finite time lengths) the range-converted time-average of the sum of the weighted inputs $\mathbb{E}\left[W_n^j\right]\mathbb{E}\left[X_n^j\right] \approx \widetilde{W}_N^j \widetilde{X}_N^j$.

B. Markov Chain Modeling

The proposed SCSD adder's long-term stochastic dynamics can be analyzed by describing the operation of the first-order SDM as a Stochastic Finite-State Machine (SFSM) and consequently modeling it using a Markov Chain (MC) [24]. To proceed, it is important to explain first 1) the MC's state space and 2) the MC's transition probabilities.

The quantizer's operation, according to (3), expresses the behavior of the SFSM as a Moore FSM, given the relation of the current output Z_n to the state T_n. The MC's states are $\mathcal{S} = \{0, 1, \ldots, M-1\}$ corresponding bijectively to those of the register, $\mathcal{T}$. The MC's current state in $\mathcal{S}$ is denoted by S_n.

The MC's transition probabilities of the MC are determined by the probability distribution of Y_n within $\mathcal{Y} = \{0, 1, \ldots, k\}$. Since Y_n is a sum of binary random variables with potentially different probability distributions, to derive the probability distribution of Y_n we use the probability generating function (P.G.F.) of it $G_{Y_n}(s) \triangleq \mathbb{E}(s^{Y_n})$, $s \in \mathbb{R}$, calculated as

$$\begin{aligned} G_{Y_n}(s) &= \mathbb{E}\left(s^{U_n^1 + \cdots + U_n^k}\right) \\ &= \mathbb{E}\left(s^{U_n^1}\right) \ldots \mathbb{E}\left(s^{U_n^k}\right) \\ &= \prod_{j=1}^{k} \Big((1 - p_{U^j}) + p_{U^j} s\Big), \end{aligned} \tag{6}$$

where in the second step we have used the fact that U_n^j are independent with respect to j and i.i.d. with respect to n. From (6) and for $r = 0, 1, \ldots, k$ it is

$$P_r(Y_n = r) = \left(\frac{1}{r!}\right) \frac{d^r}{ds^r} \left(G_{Y_n}(s)\right)\Big|_{s=0}. \tag{7}$$

The MC's state update is similar to that of the register's one in (4), since it is determined by the previous state S_{n-1}, the current input V_n and the previous output $Z_{n-1} = Q(T_{n-1})$,

$$S_n = \max\left\{0, \min\left\{S_{n-1} - Q(S_{n-1}) + V_n, M-1\right\}\right\}. \tag{8}$$

Note that S_n depends on both S_{n-1} and Z_{n-1}, which is the quantized version of S_{n-1}. This makes the analysis a little more involved. We have that $V_n \in \{-k, \ldots, k\}$ and $Z_{n-1} = Q(T_{n-1}) \in \{\pm 1\}$. The value of Z_n partitions the state set $\mathcal{S}$ into two (disjoint) subsets $\mathcal{S}_a = \{0, 1, \ldots, M/2-1\}$ and $\mathcal{S}_b = \{M/2, \ldots, M-1\}$, such that $S_n \in \mathcal{S}_a \Leftrightarrow Z_n = -1$ and $S_n \in \mathcal{S}_b \Leftrightarrow Z_n = 1$. The transition probabilities are used to build the $M \times M$ transition probability matrix $H \triangleq [P_r(S_n = \sigma_j | S_{n-1} = \sigma_i)] = [p_{\sigma_i,\sigma_j}]$, $\sigma_i, \sigma_j \in \mathcal{S}$, which is also partitioned based on $\mathcal{S}_a$ and $\mathcal{S}_b$.

The following example, with $k = 3$ inputs and $M = 8$ states, illustrates the use of the above definitions. Random variable Y_n takes values within $\{0, 1, 2, 3\}$ and $V_n = 2Y_n - k$ belongs to $\{-3, -1, 1, 3\}$. Equation (7) is used to calculate the probabilities $p_Y^r \triangleq P_r(Y_n = r)$, which are associated to the state transition from S_{n-1} to S_n:

- If $S_{n-1} \in \mathcal{S}_a = \{0, 1, \ldots, M/2-1\}$
  - If $Y_n = 0$, then $S_n = \max\{0, S_{n-1} - 2\}$
  - If $Y_n = 1$, then $S_n = S_{n-1}$,
  - If $Y_n = 2$, then $S_n = S_{n-1} + 2$,
  - If $Y_n = 3$, then $S_n = S_{n-1} + 4$.
- If $S_{n-1} \in \mathcal{S}_b = \{M/2, \ldots, M-1\}$
  - If $Y_n = 0$, then $S_n = S_{n-1} - 4$,
  - If $Y_n = 1$, then $S_n = S_{n-1} - 2$,
  - If $Y_n = 2$, then $S_n = S_{n-1}$,
  - If $Y_n = 3$, then $S_n = \min\{M-1, S_{n-1} + 2\}$.

The above result in

$$H = \begin{bmatrix} p_Y^0+p_Y^1 & 0 & p_Y^2 & 0 & p_Y^3 & 0 & 0 & 0 \\ p_Y^0 & p_Y^1 & 0 & p_Y^2 & 0 & p_Y^3 & 0 & 0 \\ p_Y^0 & 0 & p_Y^1 & 0 & p_Y^2 & 0 & p_Y^3 & 0 \\ 0 & p_Y^0 & 0 & p_Y^1 & 0 & p_Y^2 & 0 & p_Y^3 \\ p_Y^0 & 0 & p_Y^1 & 0 & p_Y^2 & 0 & p_Y^3 & 0 \\ 0 & p_Y^0 & 0 & p_Y^1 & 0 & p_Y^2 & 0 & p_Y^3 \\ 0 & 0 & p_Y^0 & 0 & p_Y^1 & 0 & p_Y^2 & p_Y^3 \\ 0 & 0 & 0 & p_Y^0 & 0 & p_Y^1 & 0 & p_Y^2 + p_Y^3 \end{bmatrix}. \quad (9)$$

The probability state vector is defined accordingly as

$$\pi_n = [P_r(S_n=0), P_r(S_n=1), \ldots, P_r(S_n=M-1)]. \quad (10)$$

Assuming that the MC's starting state S_0 can be any one within $\mathcal{S}$, then the initial distribution vector is

$$\pi_0 = \frac{1}{M}\underline{1}^T \in [0, 1]^M \quad (11)$$

where $\underline{1}$ is the column vector of M ones. It can be used along with the transition probability matrix H from (9) to calculate the states' probability distribution vector as

$$\pi_n = \pi_0 H^n \in [0, 1]^M. \quad (12)$$

The states' probability distribution vector enables the derivation of the output's first-moment statistics. The expected value of Z_n is

$$\mathbb{E}[Z_n] = \sum_{z\in\{\pm 1\}} z P_r(Z_n = z) = (-1)\pi_n e_a^T + \pi_n e_b^T, \quad (13)$$

where $e_a = \sum_{i=1}^{M/2} e_i \in \mathbb{R}^M$ and $e_b = \sum_{i=M/2+1}^{M} e_i \in \mathbb{R}^M$, with $e_i = \left[0, \ldots 0, 1, 0, \ldots, 0\right] \in \mathbb{R}^M$ being the i-th standard

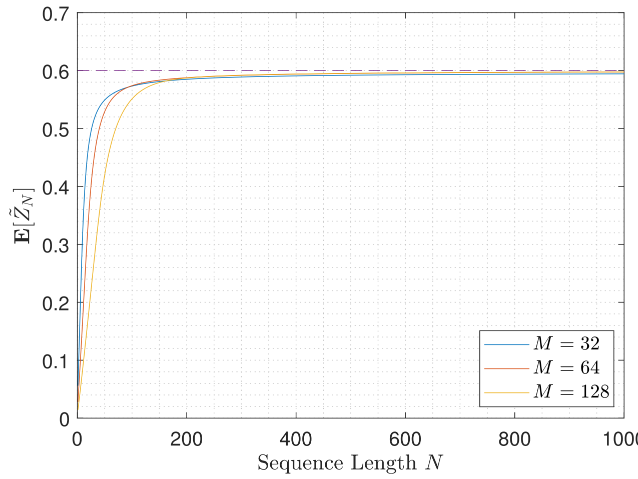

Fig. 3. Expected value of the output's time-average, $\mathbb{E}[\widetilde{Z}_N]$, calculated using (14), estimating the sums of three inputs with probability values $p_U^1 = 0.1, p_U^2 = 0.2, p_U^3 = 0.3$, as the sequence length increases $N = 1, \ldots, 1000$, for three different number of states $M = 32, 64, 128$.

vector. The result of (12) along with (13) can be used to derive the expected value of the time-average as

$$\mathbb{E}[\widetilde{Z}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[Z_n] = \frac{1}{N}\sum_{n=1}^{N}\left((-1)\pi_n e_a^T + \pi_n e_b^T\right)$$
$$= \frac{1}{N}\pi_0\left(\sum_{n=1}^{N} H^n\right)\left((-1)e_a^T + e_b^T\right). \quad (14)$$

In Fig. 3, the expected value of the time-average, $\mathbb{E}[\widetilde{Z}_N]$, is plotted using (14), for sequence length $N = 1, 2, \ldots, 1000$, number of states $M = 32, 64, 128$ and input probability values $p_U^1 = 0.1, p_U^2 = 0.2, p_U^3 = 0.3$. As the number of states M increase, $\mathbb{E}[\widetilde{Z}_N]$ approaches the input probability value 0.6 for sufficiently large N.

C. Error Characteristics

To evaluate the SCSD adder's computational performance, we use the Mean Absolute Error (MAE) defined as

$$Z_{\text{Error}} = \mathbb{E}\left|\widetilde{Z}_N - \hat{Z}\right|, \quad (15)$$

where $\hat{Z} \in [-1, 1]$ is defined as $\hat{Z} = \hat{U}^1 + \ldots \hat{U}^k$, following the definitions in Section II. We proceed with estimating the MAE numerically for $k = 2$ as follows; input values $\hat{U}^1, \hat{U}^2$ are randomly selected such that $-1 \leq \hat{U}^1 + \hat{U}^2 \leq 1$ and the simulation is conducted for 10^4 i.i.d. runs with independent sequences $\{U^1\}_{n=1}^N, \{U^2\}_{n=1}^N$. Note that here Sobol sequences [25] are considered. For input sequence length $N =$ 64-bit and register size of $m =$ 4-bits the MAE's distribution is shown in Fig. 4. The mean and standard deviation are $3.1 \cdot 10^{-2}$ and $1.2 \cdot 10^{-2}$ respectively.

To further investigate the SCSD adder's error characteristics, we calculate the MAE for different number of inputs. For this reason, we consider a number of inputs $k = 2, 4, 8, 10, 12, 14, 16$ and calculate the MAE using (15) as above, for increasing sequence lengths $N = 64, 128, 256, 512, 1024$. The results are illustrated in Fig. 5. It can be seen that when the number of inputs is relatively

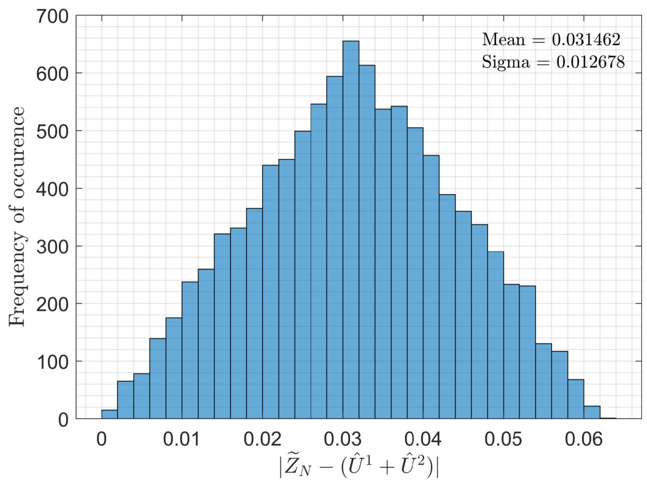

Fig. 4. MAE distribution for $k = 2$ inputs calculated using (15), for 10^4 i.i.d. runs, parametrized with sequence length $N = 64$-bit and a register size of $m = 4$-bits.

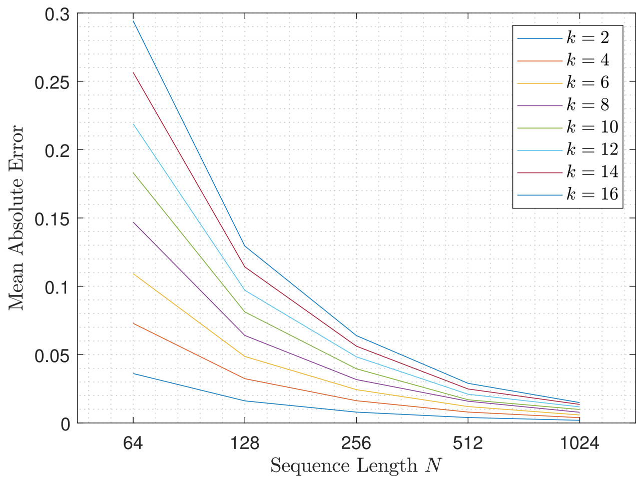

Fig. 5. MAE calculated using (15) for increasing number of inputs k and sequence lengths N. For each k, N, 10^4 i.i.d. runs are considered.

small, namely $k = 2, 4, 6$, the maximum MAE value is $1.1 \cdot 10^{-1}$ for $N = 64$-bit sequences and it is further reduced to 10^{-2} values when $N \geq 128$. On the other hand, when the number of inputs is increased to $k = 8, 10, 12, 14, 16$, the input sequence length should be $N \geq 256$ to result in MAE values with order of magnitude 10^{-2}.

D. Hardware Resources

The hardware resources of the proposed SCSD adder are compared to existing ones from the SC literature. All adder architectures are described using Verilog HDL and then their designs are fed to the Synopsys Design Compiler so as for their hardware resources to be extracted using the FreePDK CMOS library at $45nm$ [26]. The following estimates are provided: 1) the total area in μm^2, 2) the average power consumption for the maximum operating frequency in mW, 3) the delay (critical path) in ns and 4) the energy defined as the power $\times$ delay product in pJ. In Table I the results per clock cycle are cited. Note that in the comparisons, only two-input adders are considered.

From Table I it can be seen that compared to the MUX adder utilizing a Sobol sequence generator and the ESL one

TABLE I
STOCHASTIC ADDERS HARDWARE RESOURCES COMPARISON

Adder	Register (bit)	Area (μm^2)	Power (mW)	Delay (ns)	Energy (pJ)
Proposed SCSD	$m = 3$	91.98	0.071	1.6	0.11
[10]	$m = 2$	59.60	0.053	1.4	0.074
	$m = 3$	83.49	0.077		0.108
	$m = 4$	98.30	0.084		0.117
	$m = 5$	112.61	0.098		0.137
[27]		22.41	0.021	0.8	0.016
[28]		54.39	0.040	1.2	0.048
[11]		92.49	0.071	1.2	0.085
MUX Adder Sobol seq. generator size b	$b = 4$	109.34	0.241	0.8	0.192
	$b = 5$	117.32	0.272		0.216
	$b = 6$	125.03	0.325		0.261
	$b = 7$	140.19	0.362		0.289
	$b = 8$	164.83	0.378		0.302
	$b = 9$	188.54	0.440		0.352
	$b = 10$	201.61	0.461		0.368
[21] Sobol seq. generator size b	$b = 4$	113.48	0.251	0.8	0.200
	$b = 5$	119.69	0.282		0.225
	$b = 6$	136.24	0.330		0.264
	$b = 7$	144.54	0.372		0.297
	$b = 8$	164.61	0.380		0.304
	$b = 9$	188.64	0.445		0.356
	$b = 10$	204.75	0.466		0.372

in [21], the SCSD adder results in reduced area, power and energy consumption due to the additional b-bit SNG that the former ones use. Compared to the adder in [11], the SCSD adder results in similar hardware resources with slightly higher energy due to its higher critical path. Compared to the adder in [10] with internal register size $m = 4, 5$-bits, the proposed SCSD adder occupies reduced resources. However, in the design of adders trees, the adder in [10] will occupy more resources than the proposed SCSD adder, as the register cost is proportional to $2-5$-bits multiplied by the number of adders used in the realization of the adder tree. Compared to the adders in [27], [28], the proposed SCSD results in increased hardware resources, but, from a design perspective the adders in [27], [28] are subject to the weaknesses introduced by scaling adders when forming adder trees, including resolution drop, constraining cascaded operations etc. [10]. Moreover, the adders in [11] and [21] extend the design challenges of adder trees as they operate on different SC number representations impacting on the hardware resources [10]. On the other hand, the proposed SCSD adder's advantages, eliminates the design challenges introduced by the above adders in the design of SC adder-trees and multiply-and-add operations.

IV. SCSD ADDER & NEURAL NETWORK DESIGN

In this section we show how the proposed SCSD adder can be used as a basic building block to realize a SC neuron and to form a SC MLP.

A. SCSD Adder Artificial Neuron

The proposed SCSD adder's binary output allows for further processing using SC-based logic gates and/or SFSMs. The

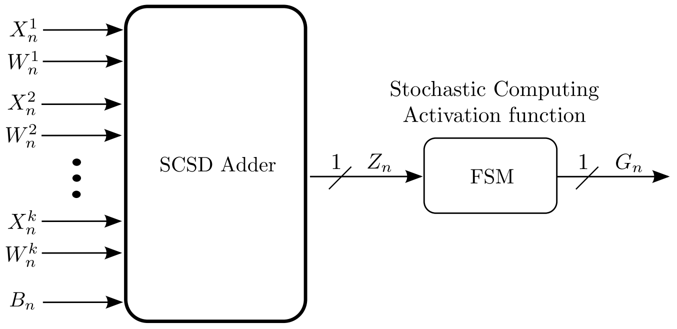

Fig. 6. SC neuron realized using the proposed SCSD adder architecture shown in Fig. 1. The non-linear activation function is realized using a single-bit input/output Stochastic Finite-State Machine.

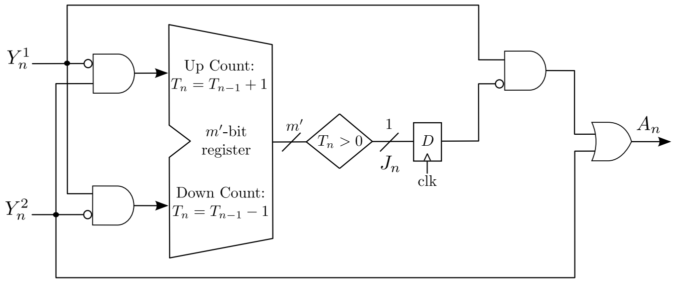

Fig. 7. Stochastic MAX architecture proposed in [32] realizing the clipped ReLU function according to (16).

SFSMs can approximate non-linear functions, within the context of SC, including activation ones [16]. They have a set of states, and the current state is updated according to the current bit-value of the input sequence. Their are realized efficiently using saturating up/down counters, outputting a logic 0 or 1 which is a function of the current state [16]. Therefore, placing a SFSM after the proposed SCSD adder allows us to build an artificial neuron, as shown in Fig. 6. Note that the sequence $\{B_n\}_{n=1}^N$ refers to the bias that can be optionally used, which is added along with $\{U_n^j\}_{n=1}^N$ in the architecture of Fig. 1.

The SCSD is designed so that the expected value of its output $\hat{G}$ is given by $\hat{G} = \phi(\hat{Z})$ where ϕ is the desirable neuron activation function. Approximately it is also the case that $\widetilde{G}_N \approxeq \phi\left(\widetilde{Z}_N\right)$. Among the widely used activation functions in SC, including the hyperbolic tangent (tanh), the exponential and the rectifier linear unit (ReLU) [16], [17], the ReLU is the most popular one in NNs [1].

For an input sequence $\{X_n\}_{n=1}^N$ with time-average value $\widetilde{X}_N$, the non-linear activation function ReLU is defined as $\text{ReLU}(\widetilde{X}_N) \approxeq \max(0, \widetilde{X}_N)$. However, since the SC's range is constrained in $[-1, 1]$, the clipped ReLU below is preferred which is a variation of the ReLU, i.e.

$$\text{Clipped ReLU}\left(\widetilde{X}_N\right) \approxeq \min\Bigg(\max\left(0, \widetilde{X}_N\right), 1\Bigg). \quad (16)$$

It is apparent that architectures realizing the max function are of high importance, with several architectures being explored in SC [17], [29], [30], [31], [32], [33]. Among them, we choose the one proposed in [32], due to its property of combining short-sequence lengths with high computational accuracy. The max architecture from [32] is shown in Fig. 7.

To proceed with a brief explanation of the max architecture's operation principle, we assume that its two sequences $\{Y_n^1\}_{n=1}^N, \{Y_n^2\}_{n=1}^N$ are i.i.d., while $\{A_n\}_{n=1}^N$ is the output

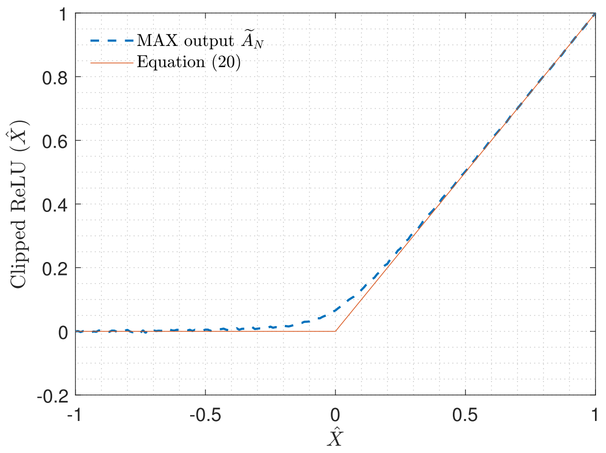

Fig. 8. Approximating the clipped ReLU of (16) using the Stochastic MAX architecture of Fig. 7, for input values $\hat{X} \in [-1, 1]$, with 10^3 i.i.d. runs on each input value, sequence length $N = 256$ and register size $m' = 4$-bits.

sequence. The m'-bit register's purpose is to count the signed-bit differences between the current bits of the two inputs; the register's current value is increased by 1-bit if $Y_n^1 > Y_n^2$, decreased by 1-bit if $Y_n^1 < Y_n^2$ and maintained otherwise.

With initial state $T_0 = 0$, the register's current state T_n is up and down counted within $\mathcal{T}_M = \{0, 1, \ldots, M'-1\}$, where $M' = 2^{m'}$ is the number of states. Hence, the state update process is described as $T_n = \max\left\{\min\left\{T_{n-1} + \overline{Y}_n^1 Y_n - Y_n^1 \overline{Y}_n^2, M'-1\right\}, 0\right\}$, where $\overline{Y}_n^j = 1 - Y_n^j$, $j = 1, 2$. Moreover, using $J_n \triangleq T_n > 0$ and by inspecting the architecture of Fig. 7, the output is determined as $A_n = \text{OR}(Y_n^2, \text{AND}(Y_n^1, \overline{J_{n-1}}))$. For further analysis of the stochastic MAX architecture the reader is referred to [32].

To showcase the effectiveness of MAX architecture of Fig. 7 in approximating the clipped ReLU function of (16), we proceed as follows; we select $2 \cdot 10^2$ uniformly distributed input values $\hat{X} \in [-1, 1]$ and for 10^3 i.i.d. runs on each input value we calculate the mean of the output's time-average $\widetilde{A}_N$. For input sequence length $N = 256$-bits and a register size of $m' = 4$-bits, the results are illustrated in Fig. 8. It is observed the MAX's output yields satisfactory results, with a small error for small absolute values of $\hat{X}$.

B. Forming a SC Multi-Layer Perceptron

The SCSD neuron in Fig. 6 can be used as a basic building block to form a SC-based MLP, which is referred to as SCSD MLP. The MLP belongs to a class of feed-forward NNs, with the network's architecture consisting mainly of three types of layers; input, hidden and output. In each layer, every unit is connected to every other one in the next layer, meaning that the MLP is fully connected. The layers within the MLP's architecture are shown in Fig. 9 and are discussed bellow.

- *Input Layer:* The units of the input layer are the values of the input features, which in our case have been converted into sequences $\{X_n^j\}_{n=1}^N$, $j = 1, 2, \ldots, k$ by SNGs. The same applies to the values of the weights existing in all layers. For the sequence generation, Sobol number generators are considered as they require shorter sequence lengths when compared to the LFSR number generators

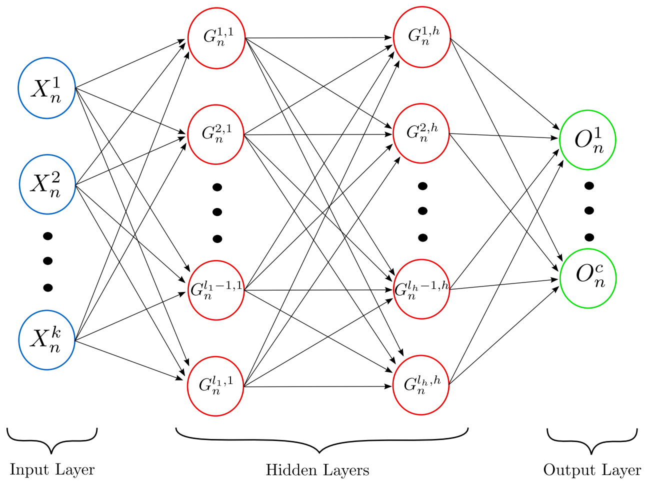

Fig. 9. Multi-Layer Perceptron network architecture. Each hidden layer is realized using the proposed SC neuron of Fig. 6 containing the proposed SCSD adder architecture of Fig. 1.

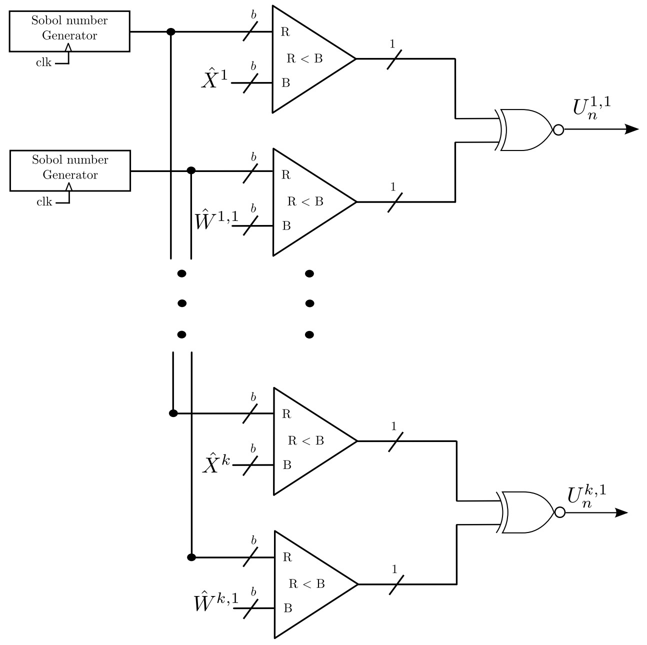

Fig. 10. Example of sequence generation in the input layer, with $\hat{X}^j, \hat{W}^{j,1}, j = 1, \ldots, k$ corresponding to the values of the inputs and the weights of a single neuron respectively. The Sobol number generators are shared among the inputs and the weights respectively.

when approximating a binary number with a stochastic one [25]. Moreover, they can be shared among the inputs and the weights respectively. An example of the sequence generation for a single neuron in the input layer is shown in Fig. 10.

- *Hidden Layers:* Each hidden layer is formed by stacking l_h neurons in parallel and the number of layers can be of any depth h. As such, considering the SCSD-based neuron of Fig. 6 the time-average output of each neuron in the MLP of Fig. 9 is described as

$$\widetilde{G}_N^{\lambda_\eta,\eta} \approx \phi\left(\widetilde{Z}_N^{\lambda_\eta,\eta}\right) \approx \min\left(\max\left(0, \widetilde{Z}_N^{\lambda_\eta,\eta}\right), 1\right), \quad (17)$$

where the clipped ReLU from (16) is used, $\lambda_\eta = 1, 2, \ldots, l_\eta$ and $\eta = 1, 2, \ldots, h$ denote the current unit in each layer and the current hidden layer respectively.

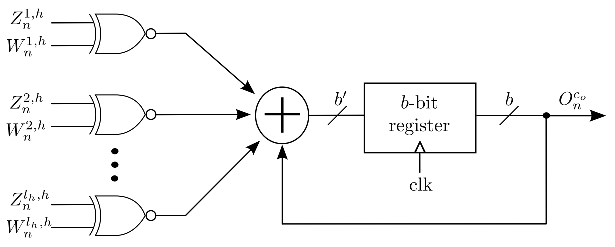

Fig. 11. A multiply-and-accumulate processing block realizing each unit $O_n^{c_o}$ existing in the output layer. The result is obtained after N clock cycles.

- *Output Layer:* The MLP's output layer, obtains the desired predictions and is of length c, corresponding to the number of classes being learned. The output layer is the last processing stage within the MLP, so the output of each neuron is realized in SC using a multiply-and-accumulate unit as shown in Fig. 11, with the result obtained after N clock cycles. In addition, the following should be noted: 1) the bit-width b' of the multiply-and-accumulate unit is determined by the number of the inputs to each unit in the output layer, l_h, and 2) the register size should be such that $N = 2^b$, according to the definitions in Section II.

V. SCSD MLP Performance

The performance of the SCSD MLP is demonstrated with multiclass classification using the MNIST dataset [34]. It consists of 60, 000 training samples and 10, 000 testing samples of grayscale images with pixel size 28×28. Each image represents handwritten digits with values ranging from 0 to 9, resulting in 10 classes in total. In the experiments, two network architectures are considered, set to $784 - 100/200 - 10$; a 784 input layer, a hidden layer of 100/200 neurons and an output layer with 10 neurons corresponding to the dataset's classes.

With respect to the training procedure, in the hidden layers neurons employ the clipped ReLU defined in (16), whereas in the output layer the *softmax* activation function is used for the classification. The values of the inputs and the weights in all layers are constrained to range $[-1, 1]$ so as to be processed in the SC domain during the inference phase. Moreover, the use of bias in all layers is not considered as it would further tax on the hardware resources due to sequence generation. The training procedure for the weights' extraction is conducted using Python and the keras library.

Once the values of the weights are extracted from the training phase, they are used in the MLP for the inference (testing) procedure. For the evaluation of the MLP's inference along with the SCSD-based realization using different sequence lengths $N = 256, 512, 1024$, fixed point (FxP) using 6, 8, 16 bits and Floating Point (FP) using 32 bits number representations are considered. The inference procedure of all MLPs is done using MATLAB.

A. Inference Accuracy

The classification performance of the MLP is evaluated using the accuracy metric, which is the ratio of the number of correct predictions to the total number of predictions. In the MNIST case, the total number of predictions corresponds to the number of testing samples, equal to 10, 000. To derive the

TABLE II
INFERENCE ACCURACY IN PERCENTAGES (%) OF THE PROPOSED SCSD, FXP AND FP MLP REALIZATIONS

	MLP 784-100-10	MLP 784-200-10
Proposed SCSD	(mean ± std)	(mean ± std)
Seq. Length $N = 256$-bit	91.25 ± 1.38	92.04 ± 1.08
Seq. Length $N = 512$-bit	95.41 ± 0.56	96.23 ± 0.68
Seq. Length $N = 1024$-bit	95.94 ± 0.41	96.68 ± 0.40
FxP 6-bit	95.18	95.89
FxP 8-bit	95.45	96.24
FxP 16-bit	95.67	96.35
FP 32-bit	96.68	97.43

TABLE III
HARDWARE RESOURCES REQUIRED FOR THE REALIZATION OF A 784-INPUT NEURON

	Area (μm^2)	Power (mW)	Delay (ns)	Energy (pJ)	ADP ($\times 10^3$) ($\mu m^2 \times ns$)	EDP ($pJ \times ns$)
SCSD	39976.68	4.06	4	16.24	159.91	64.96
FxP 6-bit	339000.79	3.99	4.2	16.75	1423.80	70.38
FxP 8-bit	703907	6.18	4.2	25.95	2956.40	109.02
FxP 16-bit	2368023.5	12.5	4.2	52.5	9945.70	220.5

accuracy of the SCSD MLP, the mean and standard deviation (std) of 10 independent runs over the testing samples are considered. This is due to the presence of the LFSR-based SNG existing in the MAX architecture where the input is compared to the value 0.5 (bipolar format of the value 0) so as to realize the clipped ReLU from (16). Sobol SNGs are reported to reduce the approximation accuracy of SFSMs over LFSR SNGs and therefore the latter ones are preferred [25]. The accuracy results of the proposed SCSD MLP accompanied by the FxP and FP realizations in percentages are cited in Table II.

From the results shown in Table II, it is observed that for the proposed SCSD MLP, when the sequence length N increases from 256-bit to 1024-bit length the std decreases, which is due to the improvement of the sequences' convergence in both network architectures. In addition, when the number of hidden layers is increased from 100 to 200, the percentage accuracy is improved by 0.79, 0.82 and by 0.74 for sequence lengths $N = 256, 512$ and 1024-bits respectively. It should be noted that experiments using $N <= 128$-bit sequence lengths were also considered, but, resulted in accuracy values $< 90\%$.

Compared to the FxP 6-bit realization, the SCSD MLP results in better percentage accuracy results when the sequence length $N = 512, 1024$, with values corresponding to an increase by 0.23 and 0.76 for the network $784-100-10$ and by 0.34 and 0.79 for the network $784-200-10$ respectively. Compared to the FxP 8-bit realization it can be seen that the SCSD MLP with sequence length $N = 512$-bit achieves similar percentage accuracy results, but, when the sequence length increases to $N = 1024$, the percentage accuracy of the SCSD MLP is increased by 0.49 and 0.44 for the networks $784-100-10$ and $784-200-10$ respectively. Compared to the FxP 16-bit realization, the percentage accuracy of the SCSD MLP for the networks $784-100-10$ and $784-200-10$ is increased by 0.27 and 0.33 respectively using sequence length $N = 1024$-bits, but, it is decreased by 0.26 and 0.12 respectively using sequence length $N = 512$-bits. On the other hand, compared to the FP MLP realization, an expected reduction of approximately 1% in the percentage accuracy is observed for the SCSD realizations, which is slightly higher for the FxP realizations.

B. Hardware Resources

The largest computational block within the MLP is the input layer considering its 784 inputs, making reasonable to investigate on its hardware resources. To this end, the SCSD and FxP neurons are described first using Verilog HDL and then they are fed into the Synopsys Design Compiler so as to extract their hardware resources using the FreePDK CMOS library at $45nm$ [26]. The following estimates are provided: 1) the total area in μm^2, 2) the average power consumption for the maximum operating frequency in mW, 3) the delay (critical path) in ns, 4) the energy per clock cycle in pJ, defined as the average power × delay product, 5) the area-delay product (ADP) in $\mu m^2 \times ns$ and 6) the energy-delay product (EDP) in $pJ \times ns$. The results for the hardware resources are cited in Table III. It should be noted that the energy, the ADP and the EDP results for the proposed SCSD neuron presented in Table III refer to a single clock cycle.

According to Table III it can be seen that the SCSD neuron reduces the area by 88.20%, 94.32% and 98.31% of the 6-bit, 8-bit and 16-bit FxP ones respectively, resulting to ×8.48 ×17.60 and ×59.23 smaller area respectively. The ADP results follow the same direction as the area ones given that the delay between the SCSD and FxP neurons is similar, meaning that the SCSD neuron reduces by 88.76%, 94.59% and 98.40% the ADP of the 6-bit, 8-bit and 16-bit FxP neurons respectively. With respect to the energy per operation, the SCSD reduces the energy by 3.09%, 37.43% and 69.06% of the of the 6-bit, 8-bit and 16-bit FxP realizations respectively, whereas the EDP follows similarly with values corresponding to a reduction by 7.70%, 40.41% and 70.53% of the 6-bit, 8-bit and 16-bit FxP realizations respectively. It should be noted that the FP number representation introduces large hardware overhead when compared to the FxP number representation due to the presence of the FP multipliers [1], [19], making it less attractive for massive multiply-and-add operations. Similar large hardware overhead is reported when comparing the FP multipliers to the SC multipliers when the SNG sharing scheme is included [19]. Therefore, despite the FP MLP's accuracy improvement of approximately 1%, the hardware resources required to realize a FP neuron are not considered in Table III.

C. Related Work

For a fair comparison with the related work in SC in terms of classification performance, it is reasonable to consider the relative error of the inference accuracy given that the network architectures differ. It is defined as

$$\epsilon = \left| \frac{\text{Accuracy}_{FP} - \text{Accuracy}_{SC}}{\text{Accuracy}_{FP}} \right|, \tag{18}$$

where Accuracy_{FP} and Accuracy_{SC} are the classification accuracies using FP and SC number representations respectively. In terms of hardware efficiency, the area

TABLE IV
PERFORMANCE COMPARISON OF SC-BASED MLPS IN INFERENCE ACCURACY AND HARDWARE RESOURCES EFFICIENCY FOR THE REALIZATION OF THE COMPUTATIONAL UNITS

Work	Architecture	Seq. Length N	Relative error (ϵ) of Accuracy on MNIST	Hardware Efficiency (as a % of the FxP)	
				Area	Energy
Proposed SCSD	784-100-10 784-200-10	512/1024 512/1024	0.0131/0.0077 0.0123/0.0077	5.6% / 1.6% (FxP 8/16-bit)	62.56% / 30.93% (FxP 8/16-bit)
[15]	784-100-200-10	1024	0.0018*	50% (FxP 9-bit)	30% (FxP 9-bit)
[19]	784-200-100-10	4096	0.0133	40.7% (FxP 8-bit)	38% (FxP 8-bit)
[20]	784-128-128-10	1/gradient	0.0179**	7.3% (FxP 10-bit)	10% (FxP 10-bit)

* The highest accuracy score is used to calculate the relative error, taken from [15]
** The relative error of accuracy corresponds to the training procedure

occupation and energy consumption in percentages over that of the FxP arithmetic are reported. The performance results are cited in Table IV.

In [15], a stochastic neuron is realized using an APC, followed by a FSM operating as the non-linear function *tanh* (BTanh) and implemented as binary up/down counter. The design of the BTanh is fixed, in the sense that the FSM's number of states affecting the *tanh*'s approximation are derived using numerical experiments for specific input sequence lengths. Moreover, the input sequence's bit-length driving the FSM affecting the number of the FSM's states is not considered.

According to Table IV, the relative accuracy error of the MLP in [15] is small, but, note that only the best score is reported, whereas in the proposed work and the rest ones, the average accuracy over independent runs is considered. For the hardware resources, in [15] a 200-input neuron is reported, utilizing 50% area and 30% energy of the FxP 9-bit realization On the other hand, the proposed 784-input neuron occupies only the 5.6%/1.6% of the 8/16-bit FxP's area respectively, while the energy is 62.56%/30.93% of the 8/16-bit FxP respectively. Moreover, the proposed SCSD adder opens the SC design space as it allows the use of single-bit output SFSMs, which consequently enables the realization of multiplications using AND/XNOR gates according to the SC number format used.

In [19], stochastic neurons in the input layer are realized by adopting the extended stochastic logic (ESL) [21], whereas in the rest layers they are realized using APCs. Combining two different multiply-and-add processing methods allows the use of ESL-based backpropagation circuits, enabling online training. The ESL adder tree, however, requires a TMR binary search divider, resulting in large sequence lengths for its computation and stabilization phases [1], [19].

From the results in Table IV, the relative accuracy error of the MLP in [19] using $N = 4096$-bit sequence lengths is similar to that of the proposed SCSD one when $N = 512$-bit sequence lengths are used, resulting in a $\times 8$ faster convergence for the proposed approach. Comparing the area efficiencies, it can be seen that the proposed approach results in significant savings of the FxP 8/16-bit realization, but, energy-wise the approach in [19] is better when 8-bits FxP are considered. From a design perspective, the proposed SCSD adder uses the standard SC encodings, whereas in [19] the use of ESL logic for the realization of the multiply-and-add units introduces design challenges [10].

Deviating from the previous approaches targeting multiply-and-add computational units, in [20], a signed SC gradient descent (SCGD) circuit capable of updating the value of the gradient and the weights was used in the training process of a MLP. From the computational accuracy results in [20], this method achieves significant training accuracy when compared to the FxP arithmetic with step size 2^{-10}. Note that in [20], only the FxP arithmetic is reported as an accuracy metric, hence for fair comparisons among the works cited, we consider FP arithmetic using a $784 - 128 - 128 - 10$ network, yielding 98.8% accuracy after 20 training epochs. Regarding the hardware resources, the SCGD circuits result in reduced area and energy compared to their FxP counterparts according to Table IV, but accuracy-wise the proposed approach results in smaller relative error.

VI. CONCLUSION

A stochastic computing sigma-delta adder was introduced. Its operation principle was analysed in detail, while its MC modeling resulted in the derivation of its first-moment statistics, used to demonstrate its fast convergence. The proposed SCSD adder enables the efficient realization of SC neurons as its single-bit output allows for the use of any SFSM based non-linear function. This was demonstrated with two compact MLPs comprised of SCSD-based neurons. Experimental results showed that the proposed SCSD neuron results in significant area savings, corresponding to a 94.32%/98.32% reduction with respect to 8/16-bit FxP arithmetic, achieving also acceptable computational accuracy compared to the 32-bit FP arithmetic. Finally, comparison of the proposed approach to existing ones in the SC literature showcased the advantages of the first one in terms of applicability in the SC design space.

REFERENCES

[1] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 7, pp. 2809–2824, Jul. 2021.

[2] H. Abdellatef, M. Khalil-Hani, N. Shaikh-Husin, and S. O. Ayat, "Accurate and compact convolutional neural network based on stochastic computing," *Neurocomputing*, vol. 471, pp. 31–47, Jan. 2022.

[3] C. F. Frasser et al., "Fully parallel stochastic computing hardware implementation of convolutional neural networks for edge computing applications," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 22, 2022, doi: 10.1109/TNNLS.2022.3166799.

[4] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1515–1531, Aug. 2018.

[5] N. Temenos and P. P. Sotiriadis, "Modeling a stochastic computing nonscaling adder and its application in image sharpening," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 5, pp. 2543–2547, May 2022.

[6] B. R. Gaines, "Stochastic computing systems," in *Proc. Adv. Inf. Syst. Sci.*, 1969, pp. 37–172.

[7] Y. Liu, L. Liu, F. Lombardi, and J. Han, "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 9, pp. 2213–2221, Sep. 2019.

[8] A. Morro et al., "A stochastic spiking neural network for virtual screening," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 4, pp. 1371–1375, Apr. 2018.

[9] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2925–2938, Dec. 2019.

[10] N. Temenos and P. P. Sotiriadis, "Nonscaling adders and subtracters for stochastic computing using Markov chains," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 9, pp. 1612–1623, Sep. 2021.

[11] A. Ren et al., "SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 405–418, 2016.

[12] B. Parhami and C.-H. Yeh, "Accumulative parallel counters," in *Proc. Conf. Rec. 29h Asilomar Conf. Signals, Syst. Comput.*, 1995, pp. 966–970.

[13] S. R. Faraji, M. H. Najafi, B. Li, D. J. Lilja, and K. Bazargan, "Energy-efficient convolutional neural networks with deterministic bit-stream processing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1757–1762.

[14] P.-S. Ting and J. P. Hayes, "Stochastic logic realization of matrix operations," in *Proc. 17th Euromicro Conf. Digit. Syst. Design*, Aug. 2014, pp. 356–364.

[15] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.

[16] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.

[17] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[18] Y. Liu, Y. Wang, F. Lombardi, and J. Han, "An energy-efficient online-learning stochastic computational deep belief network," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 454–465, Sep. 2018.

[19] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1273–1286, Sep. 2018.

[20] S. Liu, H. Jiang, L. Liu, and J. Han, "Gradient descent using stochastic circuits for efficient training of learning machines," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2530–2541, Nov. 2018.

[21] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rosselè, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 551–564, Mar. 2016.

[22] P. P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx MicroBlaze MCS SoC*. Hoboken, NJ, USA: Wiley, 2017.

[23] K. Hosseini and M. P. Kennedy, *Minimizing Spurious Tones in Digital Delta-Sigma Modulators*. Cham, Switzerland: Springer, 2011.

[24] N. Temenos and P. P. Sotiriadis, "A Markov chain framework for modeling the statistical properties of stochastic computing finite-state machines," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Oct. 3, 2022, doi: 10.1109/TCAD.2022.3211487.

[25] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel Sobol sequences," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1326–1339, Jul. 2018.

[26] J. E. Stine et al., "FreePDK: An open-source variation-aware design kit," in *Proc. IEEE Int. Conf. Microelectron. Syst. Educ. (MSE)*, Jun. 2007, pp. 173–174.

[27] 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 *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 13–18.

[28] P. Ting and J. P. Hayes, "Eliminating a hidden error source in stochastic circuits," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2017, pp. 1–6.

[29] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation manipulating circuits for stochastic computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1417–1422.

[30] J. Yu, K. Kim, J. Lee, and K. Choi, "Accurate and efficient stochastic computing hardware for convolutional neural networks," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 105–112.

[31] M. Lunglmayr, D. Wiesinger, and W. Haselmayr, "Design and analysis of efficient maximum/minimum circuits for stochastic computing," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 402–409, Mar. 2020.

[32] N. Temenos and P. P. Sotiriadis, "Stochastic computing max min architectures using Markov chains: Design, analysis, and implementation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 11, pp. 1813–1823, Nov. 2021.

[33] P. P. Sotiriadis and N. Temenos, "Compact MAX and MIN stochastic computing architectures," *Integration*, vol. 87, pp. 194–204, Nov. 2022.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

Nikos Temenos (Member, IEEE) received the B.Sc. degree in computer and systems engineering from the Piraeus University of Applied Sciences, Greece, in 2015, the M.Sc. degree in microelectronics from the National and Kapodistrian University of Athens, Greece, in 2017, and the Ph.D. degree in electrical and computer engineering from the National Technical University of Athens, Greece, in 2022.

He has authored several IEEE conferences. His main research interests include digital VLSI design, computer arithmetic, and algorithms and architectures for stochastic computing targeting FPGA and ASIC technologies. He is a regular reviewer for many IEEE TRANSACTIONS and conferences.

Paul P. Sotiriadis (Fellow, IEEE) received the Diploma degree in electrical and computer engineering from the National Technical University of Athens (NTUA), Greece, in 1994, the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1996, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2002.

In 2002, he joined as a Faculty Member with the Department of Electrical and Computer Engineering, Johns Hopkins University. In 2012, he joined as a Faculty Member with the Department of Electrical and Computer Engineering, NTUA. He is currently a Professor of electrical and computer engineering and the Director of the Electronics Laboratory at NTUA. He has authored and coauthored more than 200 research publications, most of them in IEEE journals and conferences, and holds one patent. He has contributed several chapters to technical books. His research interests include design, optimization, and mathematical modeling of analog, mixed-signal, and RF integrated and discrete circuits, sensor and instrumentation architectures with emphasis in biomedical instrumentation, advanced RF frequency synthesis, and the application of machine learning and general AI in the operation and design of electronic circuits. He has been a member of technical committees of many conferences. He is a Governing Board Member of the Hellenic (National) Space Center, Greece. He has received several awards, including the prestigious Guillemin-Cauer Award from the IEEE Circuits and Systems Society in 2012 and the Best Paper Award from the IEEE International Symposium on Circuits and Systems in 2007, the IEEE International Frequency Control Symposium in 2012, the IEEE International Conference on Modern Circuits and Systems Technologies in 2019, the IEEE International Conference on Microelectronics (ICM) in 2020, the IEEE International Conference on Microelectronics (ICM) in 2021, the IEEE Symposium on Integrated Circuits and Systems Design (SBCCI) in 2021, and the IEEE Circuits and Systems Society (CASS) Outstanding Technical Committee Recognition in 2022. He is an Associate Editor of the IEEE SENSORS JOURNAL. He has served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS from 2016 to 2020 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS from 2005 to 2010. He regularly reviews for many IEEE TRANSACTIONS and conferences and serves on proposal review panels.