A Stochastic Computational Multi-Layer Perceptron with Backward Propagation

Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, *Fellow, IEEE,* and Jie Han, *Senior Member, IEEE*

Abstract—Stochastic computation has recently been proposed for implementing artificial neural networks with reduced hardware and power consumption, but at a decreased accuracy and processing speed. Most existing implementations are based on pre-training such that the weights are predetermined for neurons at different layers, thus these implementations lack the ability to update the values of the network parameters. In this paper, a stochastic computational multi-layer perceptron (SC-MLP) is proposed by implementing the backward propagation algorithm for updating the layer weights. Using extended stochastic logic (ESL), a reconfigurable stochastic computational activation unit (SCAU) is designed to implement different types of activation functions such as the $tanh$ and the rectifier function. A triple modular redundancy (TMR) technique is employed for reducing the random fluctuations in stochastic computation. A probability estimator (PE) and a divider based on the TMR and a binary search algorithm are further proposed with progressive precision for reducing the required stochastic sequence length. Therefore, the latency and energy consumption of the SC-MLP are significantly reduced. The simulation results show that the proposed design is capable of implementing both the training and inference processes. For the classification of nonlinearly separable patterns, at a slight loss of accuracy by 1.32%-1.34%, the proposed design requires only 28.5%-30.1% of the area and 18.9%-23.9% of the energy consumption incurred by a design using floating point arithmetic. Compared to a fixed-point implementation, the SC-MLP consumes a smaller area (40.7%-45.5%) and a lower energy consumption (38.0%-51.0%) with a similar processing speed and a slight drop of accuracy by 0.15%-0.33%. The area and the energy consumption of the proposed design is from 80.7%-87.1% and from 71.9%-93.1%, respectively, of a binarized neural network (BNN), with a similar accuracy.

Index Terms—Stochastic computation, binary search, neural network, probability estimator, multi-layer perceptron.

1 Introduction

A MULTI-LAYER perceptron (MLP) is a type of artificial neural network (ANN) in which neurons are interconnected in several layers. It can solve problems such as the approximation (or fitting) of functions and the classification of nonlinearly separable patterns. Compared to a traditional software implementation, the hardware implementation of an ANN offers the advantages of an inherently high degree of parallelization and faster training speed. Unfortunately, a complex hardware is likely required in an MLP system, because thousands of neurons are involved in solving problems such as classification [1] [2] [3]. In contrast to a conventional binary circuit design, a stochastic computing (SC) circuit requires a low hardware complexity with a high fault tolerance to computation and soft errors [4]; such features make it feasible to implement a robust MLP at a lower hardware cost.

Stochastic neural computation was introduced in [5] [6]. Several fundamental computational elements, including adders, multipliers and squaring circuits, have been discussed for stochastic logic and a stochastic computational neural network has been built on a soft competitive learning algorithm. To reduce the area and energy consumption of a stochastic circuit, random number generators (RNGs) are shared in the generation of stochastic input sequences without affecting the computation accuracy [7]. A hardware implementation of a neural network has been developed for the radial basis function algorithm, for which multiplication is implemented using SC circuits [8]. The extended stochastic logic (ESL) is introduced into a probabilistic neural network implementation in [9]. Let the bit-width of a binary number be N; ESL expands the conventional stochastic computational domain from $[-1, +1]$ in the bipolar representation to $(-2^N - 1, 2^N + 1)$ by using two stochastic sequences to encode a real number [9]. Integral stochastic computing can also extend the computation domain of SC, but the number of required sequences increases with the range of encoded values [10]. SC circuits have also been used to implement the forward propagation in deep neural networks for character recognition [11] [12] [13]. However, the weights at different layers of the neural networks are predetermined and cannot be updated during the computation process.

In this paper, a stochastic computational multi-layer perceptron (SC-MLP) is proposed to overcome the above limitations. The proposed design utilizes an SC activation unit (SCAU) based on accumulative parallel counters (APCs) and finite state machines (FSMs). Albeit using ESL, a hybrid SC network structure is introduced for an efficient implementation. To further reduce energy consumption, the designs of a probability estimator (PE) and a stochastic divider are proposed using a triple modular redundancy

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada

- *Y. Liu, S. Liu and J. Han are with Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 1H9 Canada (email: {yidong1, siting2, jhan8}@ualberta.ca).*
- *Y. Wang is with Electrical Engineering and Computer Science Department, Syracuse University, Syracuse, NY 13244, USA (email: ywang393@syr.edu)*
- *F. Lombardi is with Department of Electrical and Computer Engineering, Northeastern University, Boston 02115, MA USA (e-mail: lombardi@ece.neu.edu).*

(TMR) and a binary search based algorithm with progressive precision. This work makes the following contributions:

- A hybrid SC network structure: In this hybrid SC-MLP structure, ESL is employed in the computation of the neurons within the input layer during the forward propagation, as well as in the gradient computation and layer weights updating during the backward propagation. The other computations are implemented by using conventional SC to reduce area and energy consumption without sacrificing classification accuracy.
- Reconfigurable activation functions: The SCAU can be reconfigured to implement different types of activation functions such as the $tanh$ and the rectifier function. The adders and subtractors in the SCAU are replaced with shift registers and comparators to further reduce circuit area and energy consumption.
- TMR-based probability estimator and divider: By utilizing a TMR voting structure in the PE and divider, the error due to stochastic fluctuations in the binary search process is significantly reduced. Therefore, the latency and energy consumption are also reduced. To the best of the authors knowledge, this is the first application of TMR and a binary search algorithm in a stochastic circuit design.
- Efficient utilization of progressive precision in SC: The operation of the perceptron is divided into a computation phase and a stabilized phase. The initial part of the stochastic sequences that carry inaccurate statistics during the computation phase is ignored, and only the latter part of the sequences that carry more accurate statistics is used in the stabilized phase. Therefore, the accuracy of the proposed design is significantly improved with a higher energy efficiency.
- Implementation of the backward propagation algorithm: A backward propagation module is designed to implement the learning algorithm in the perceptron made of hybrid SC circuits using ESL and conventional SC.

The proposed design is evaluated on the Modified National Institute of Standards and Technology (MNIST) [14] and the Street View House Numbers (SVHN) [15] datasets. It achieves similar accuracy with lower area and energy consumption compared to conventional floating and fixed-point implementations. These results show that the proposed design is advantageous for implementations of machine learning algorithms in resource-limited systems such as mobile and embedded systems.

The remainder of this paper is organized as follows. Section II provides the background on the design of a multi-layer perceptron, basic stochastic logic elements, and extended stochastic logic. Section III introduces the proposed design. Section IV shows the applications and simulation results. Section V gives the conclusion.

2 Background Review

2.1 Multi-layer Perceptron

The structure of a multi-layer perceptron (MLP) is shown in Fig. 1. The MLP includes one input layer, at least one hidden layer and one output layer. Each layer consists of multiple neurons as the fundamental computation units. One of the most widely-used learning algorithms is the backward propagation (BP) algorithm. This algorithm proceeds in two phases: the forward propagation and the backward propagation [1]. The forward propagation generates output signals based on the current inputs and layer weights. In the backward propagation, error signals are first obtained from the output signals (generated by forward propagation) with class labels of the training datasets; then the layer weights are updated using the error signals.

The BP algorithm is performed in multiple epochs. In each epoch, the perceptron is trained on the training dataset. Let $y_j^l(n)$ be the output signal of neuron j in layer l at epoch n, and $w_{ji}^l(n)$ be the layer weight between the neuron j at layer l and the neuron i in the previous layer $l-1$. The layer $l-1$ consists of m neurons. In the forward propagation, the layer weights remain unaltered and the output signals are computed on a neuron-by-neuron basis as

$$y_j^l = \phi(v_j^l(n)). \tag{1}$$

where ϕ is the activation function and $v_j^l(n)$ is defined as

$$v_j^l(n) = \sum_{i=1}^{m} w_{ji}^l y_i^{l-1}(n). \tag{2}$$

The hyperbolic tangent function ($tanh$) is a commonly implemented activation function in SC for solving non-linear classification problems [5]. It is defined as

$$\phi(v_j(n)) = tanh(2v_j(n)). \tag{3}$$

The rectifier linear unit (ReLU) function is another widely used activation function [16]. In this paper, the range of the output signal is limited to $[0, 1]$ for SC representation. The function of the clamped ReLU is defined as

$$\phi(v_j(n)) = min(1, max(0, v_j(n))). \tag{4}$$

In the backward propagation, the layer weight $w_{ji}^l(n)$ is adjusted on a layer-by-layer basis as

$$w_{ji}^l(n+1) = w_{ji}^l(n) + \eta \Delta w_{ji}^l(n). \tag{5}$$

where

$$\Delta w_{ji}^l(n) = \delta_j^l(n) \cdot y_i^{l-1}(n). \tag{6}$$

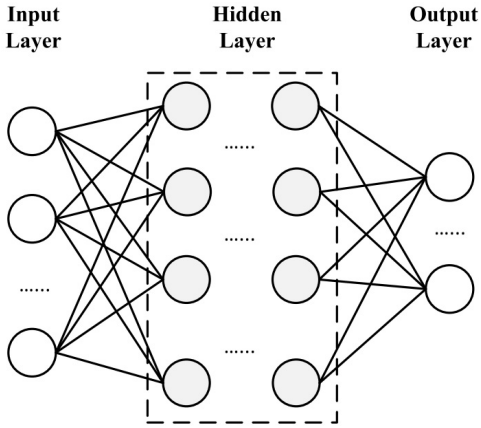


Fig. 1. Structure of the multi-layer perceptron (MLP).

η is the learning rate, initialized to 0.01, and $\delta_j^l(n)$ is the local gradient, defined as

$$\delta_j^l(n) = \begin{cases} (d_j(n) - o_j^l)\phi'(v_j^l(n)), & output\,layer; \\ \phi'(v_j^l(n)) \sum_{i=1}^{p} \delta_i^{l+1} w_{ij}^{l+1}(n), & hidden\,layer. \end{cases} \tag{7}$$

In (7), if the l^{th} layer is the output layer, o_j^l is equivalent to y_j^l in (1). $d_j(n)$ is the j^{th} variable in the class label of a training sample, so $d_j(n) - o_j^l$ indicates the error signal generated by the perceptron based on the current layer weights. However, if the l^{th} layer is a hidden layer and that the $(l+1)^{th}$ layer consists of p neurons, the local gradient is determined by the sum-of-products of δ_i^{l+1} and w_{ij}^{l+1}. In both cases, $\phi'(v_j^l(n))$ is the derivative of the activation function with respect to $v_j^l(n)$. The signal-flow graphs of the backward propagation neurons at different layers are shown in Fig. 2.

After updating the layer weights in the backward propagation, the forward propagation is repeated to generate the new error signals for the next loop and the error signals are expected to be reduced at each iteration. The forward and backward propagation processes are repeated until the maximum allowed number of epochs is reached, or the early stopping is reported [17].

Batch normalization has been used to accelerate the convergence by fixing the means and variances of the layer inputs [18]. For a layer with a d-dimensional input $x = (x^{(1)}, x^{(2)}...x^{(d)})$, each dimension is normalized independently by

$$n(x^{(k)}) = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{var[x^{(k)}]}},\ k = 1, 2...d. \tag{8}$$

As per (8), the mean of the inputs is normalized to zero and the variance to 1. In the SC-MLP, the batch normalization is utilized in the input layer but eliminated in the other layers to keep the computation within $[-1, +1]$ for SC implementation.

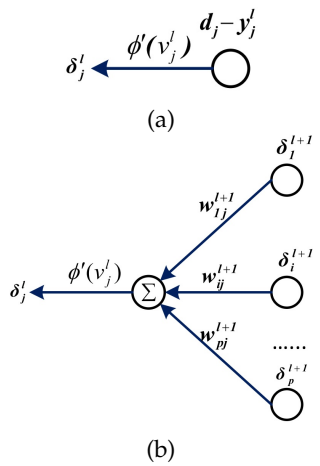


Fig. 2. Signal-flow graphs of the backward propagation process in (a) the output layer neuron and (b) the hidden layer neuron.

2.2 Binarized neural networks (BNNs)

Binarized neural networks (BNNs) have been proposed to obtain a tradeoff between accuracy and energy consumption [19]. In a BNN, the layer weights and the intermediate computation results are converted from a real value to $+1$ or -1 by either a deterministic binarization function:

$$b(w) = sign(w) = \begin{cases} +1 & if\ w \geq 0 \\ -1 & otherwise \end{cases}, \tag{9}$$

or a stochastic binarization function:

$$\hat{b}(w) = \begin{cases} +1 & with\ probability\ p = \sigma(w) \\ -1 & with\ probability\ 1 - p \end{cases}, \tag{10}$$

where

$$\sigma(x) = max(0, min(1, \frac{x+1}{2})). \tag{11}$$

After the binarization, the multiplications in (2) (5) (6) and (7) can be eliminated by utilizing XNOR operations.

In general, there are two types of binarized neural networks, the BinaryConnect [20] and the BinaryNet [19]. In the BinaryConnect algorithm, the layer weights are copied and binarized. The binarized layer weights are then used in the forward propagation (as per (1) and (2)), while the original real-valued weights are used and updated in the backward propagation (as per (5), (6) and (7)). In the BinaryNet algorithm, the $y_j^l(n)$ in (1) is binarized and used in (2) with the binarized layer weights to compute the outputs in the next layer during the forward propagation. The binarized $y_j^l(n)$ and the binarized layer weights are used in (7) to compute the local gradient, while the original real-valued parameters are used in (5) for updating the layer weights.

In inference, the binarized output of a neuron, a_n^b, is computed by

$$a_n^b = b(n(x)), \tag{12}$$

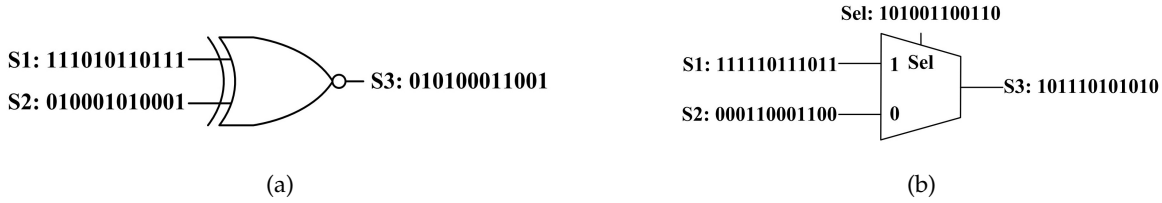
where x is the sum-of-product of the input data and the layer weights of the neuron. $n(x)$ is the batch normalized result following (8). Applying (8) and (9) into (12), the binarized output of neuron n is given by

$$\begin{aligned} a_n^b &= sign(\frac{x - E(x)}{\sqrt{var(x)}}) \\ &= sign(x - E(x)). \end{aligned} \tag{13}$$

Similar to [21], the computation of batch normalization and binarization can be implemented by using comparators, without actually computing $var(x)$ in inference. However, as the local gradient (7) is not binarized, the batch normalization is usually computed in the training process.

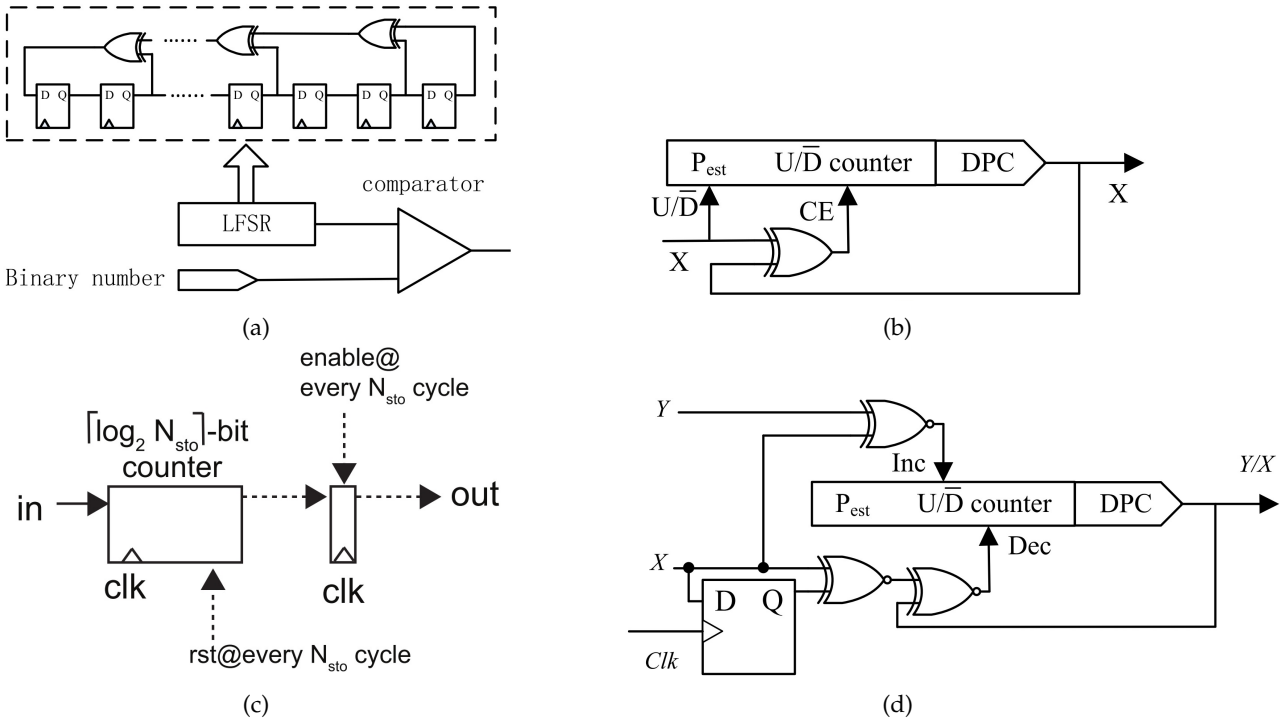
2.3 Stochastic logic elements

In stochastic computation, the presence of p 1's in a random binary bit stream with length q encodes the value p/q in the unipolar representation, or the value $(2p - q)/q$ in the bipolar representation. Therefore, a stochastic sequence represents a real number in $[0, 1]$ in the unipolar representation, or a number in $[-1, +1]$ in the bipolar representation. Stochastic computation is generally executed on a bit-wise basis for both combinational and sequential circuits. It significantly reduces the complexity of an arithmetic circuit;



(a) (b)

Fig. 3. (a) A bipolar stochastic multiplier: $P(S1) = 1/2, P(S2) = -1/3$, and $P(S3) = P(S1) \cdot P(S2) = -1/6$. (b) A stochastic adder: $P(S1) = 2/3, P(S2) = -1/3$, and $P(S3) = 0.5 \times (P(S1) + P(S2)) = 1/6$.



(a) (b) (c) (d)

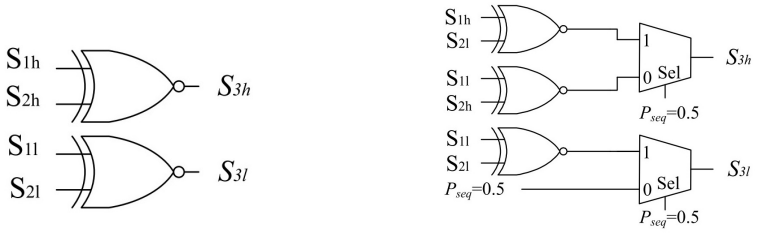
Fig. 4. (a) A digital-to-probability converter (DPC), (b) a conventional probability estimator (PE) [5], (c) a PE using flip-flops, and (d) a conventional bipolar stochastic divider [5].

for example, an $XNOR$ gate computes the product of two numbers in the bipolar representation (Fig. 3 (a)). Moreover, a stochastic adder is implemented by a multiplexer with the select signal encoding a probability of 0.5 (Fig. 3 (b)) [5].

A digital-to-probability converter (DPC) is commonly used to convert a real number into a stochastic sequence [22] [23]. It consists of a linear feedback shift register (LFSR) and a comparator (Fig. 4 (a)). Another important component used in SC is the probability estimator (PE); the PE determines the probability encoded by a specific stochastic sequence. A conventional PE (Fig. 4 (b)) is implemented by a counter and a DPC [5]. The conventional PE is based on the gradient descent algorithm. It first generates a stochastic sequence using the DPC and then uses an up/down counter to compare the probabilities encoded in the generated and input sequences. Subsequently, the PE adjusts the probability encoded in the generated sequence by increasing (or decreasing) the value in the counter if the probability encoded in the generated sequence is smaller (or larger) than the value encoded in the input sequence. When the same probability is obtained for the input and generated sequences, the probability estimator records the binary value in the counter to estimate the probability (P_{est}) encoded in the input sequence.

A different implementation of a PE using flip-flops is introduced in [24]. In Fig. 4 (c), N_{sto} is the number of clock cycles (equal to the sequence length) required for stochastic computation. In each cycle, the flip-flop based PE counts the number of 1's in the input sequence to estimate the probability. This PE requires smaller hardware than a conventional PE. However, it needs a total of N_{sto} cycles for computation.

Fig. 4 (d) shows the implementation of a conventional bipolar stochastic divider; this design is based on the same gradient descent algorithm used in a conventional PE. In general, a conventional PE incurs a high latency when the input probability is substantially different from the initial probability of the generated sequence; this feature significantly decreases the speed of computation. A stepped velocity algorithm is introduced in [5] to address this problem. In the stepped velocity algorithm, the value of the counter is processed in multiple steps, starting with 2^{N-1} and 2^{N-2} as the step size, where N is the bit width of the counter. Each time, the step size is decreased by half until it reaches



(a) (b)

Fig. 5. (a) An extended stochastic logic (ESL) multiplier, with $P(S1) \cdot P(S2) = \frac{P(S_{1h})}{P(S_{1l})} \cdot \frac{P(S_{2h})}{P(S_{2l})} = \frac{P(S_{3h})}{P(S_{3l})} = P(S3)$ [9], and (b) an ESL adder, with $P(S1) + P(S2) = \frac{P(S_{1h})}{P(S_{1l})} + \frac{P(S_{2h})}{P(S_{2l})} = \frac{P(S_{1h}) \cdot P(S_{2l}) + P(S_{1l}) \cdot P(S_{2h})}{P(S_{1l}) \cdot P(S_{2l}) + 0} = \frac{P(S_{3h})}{P(S_{3l})} = P(S3)$ [9].

1. This binary search based algorithm significantly reduces the sequence length, however the errors due to random fluctuations in the stochastic sequences may lead to an incorrect search direction. If an incorrect direction occurs in an early step, it will result in a considerable loss of accuracy. Therefore, a new design using TMR and a binary search based PE is proposed to shorten the sequence length and overcome the accuracy loss in the stepped velocity algorithm (as discussed in Section III).

2.4 Extended stochastic logic (ESL)

Stochastic computation is limited to values within $[-1, 1]$ in the bipolar representation, or $[0, 1]$ in the unipolar representation. This limitation restricts the use of stochastic designs in neural networks because the computed results can exceed the range, resulting in accuracy loss. To overcome this drawback, the extended stochastic logic (ESL) is used to expand the range of stochastic computation to $(-2^N + 1, 2^N - 1)$ for a binary value in N bits [9].

In ESL, a real number is encoded as the ratio of two stochastic sequences using the bipolar representation. Assume that the bipolar representation of the dividend sequence is p_h and the divisor sequence is p_l; then an arbitrary real number x is approximately given by the quotient as [9].

$$x = \frac{p_h}{p_l}. \tag{14}$$

The ESL multiplier and adder are shown in Fig. 5 [9].

3 Stochastic computational multi-layer perceptron (SC-MLP) design

3.1 Overall design

The proposed design of the SC-MLP circuit consists of five components: the data RAM, the forward propagation component, the backward propagation component, the layer weight register and the output register (Fig. 6).

The xData and tData RAM store the input dataset and the class labels. The forward propagation component generates output signals based on the current datasets and layer weights in the training and inference processes. The backward propagation component generates the error signals by comparing the output signals with the desired class labels; then it adjusts the layer weights in the training process. The layer weight register stores the updated values of the layer weights and loads the values into the forward and backward propagation components in the next epoch. The classification results are stored in the output register for accuracy evaluation.

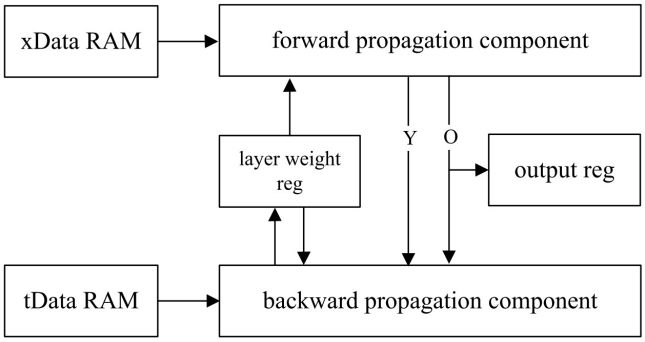


Fig. 6. The SC-MLP network. The xData RAM stores the input data, the tData RAM stores the class labels, the layer weight reg stores the weights and the output reg stores the classification results generated by the forward propagation component.

3.2 Training algorithm

The pseudo code of the SC-MLP training process is listed as following.

Code 1: Training of an SC-MLP. $\phi(\cdot)$ is the SC activation function, η is the learning rate and L is the number of layers. The function $ToESL(\cdot)$ specifies how to generate ESL sequences based on binary or conventional SC sequences. $ToConv(\cdot)$ specifies how to generate the conventional SC sequences. $clamp(k,\ a,\ b)$ specifies how to restrict the parameter k into the given range $[a,\ b]$. $Grad(\cdot)$ specifies how to compute the gradient and $Update(\cdot)$ specifies how to update the layer weights. $ToBinary(\cdot)$ specifies how to convert ESL sequences into conventional SC sequences then binary values.

Inputs: the input data a and the label d, the layer weight W^t at epoch t and the learning rate η.

Outputs: updated weights W^{t+1} at epoch $t + 1$.

{1. Forward propagation}

for $i = 1$ **to** L

  if $i == 1$

    $v_i = ToConv(sum(ToESL(a) \cdot ToESL(W_i^t)))$;

$y_i = \phi(v_i)$;
else
$y_i = \phi(sum(y_{i-1} \cdot ToConv(W_i^t)))$;
end
{2. Backward propagation}
{2.1 Compute gradient}
for $i = L$ **downto** 1
$y_{s_i} = ToESL(y_i)$;
$v_{s_i} = ToESL(v_i)$;
if $i == L$
$g_i = Grad(ToESL(d),\ y_{s_i},\ \frac{\partial \phi}{\partial v_{s_i}})$;
else
$g_i = Grad(g_{i+1},\ ToESL(W_{i+1}^t),\ \frac{\partial \phi}{\partial v_{s_i}})$;
end
{2.2 Update the layer weights}
for $i = L$ **downto** 1
$W_i^{t+1} = Update(ToESL(W_i^t),\ g_i,\ y_{s_i-1},\ \eta)$;
$W_i^{t+1} = clamp(ToBinary(W_i^{t+1}),\ -1,\ 1)$;
end

During the forward propagation, as the range of input data is not limited in $[-1, +1]$ after the batch normalization, the input data and the layer weights are converted into ESL sequences to compute the sum-of-products in (2). Then the results are converted to conventional SC sequences and sent to the activation circuit. Because the output signals are restricted in $[-1, +1]$ by $tanh$ or $[0, +1]$ by the clamped ReLU, conventional SC sequences are used in the other layers. The batch normalization is only used for the input dataset, because the outputs of the activation functions can be shifted out of the conventional SC range due to small variances, causing a loss of accuracy in the next step of the computation.

During the backward propagation, the SC-MLP uses ESL sequences to ensure that the gradients are not limited by the range. The layer weights are updated by the ESL sequences to increase accuracy. Once updated, the layer weights are first converted into conventional SC sequences and then into binary values. By this conversion, the layer weights are brought back to $[-1, +1]$ in the bipolar representation and at the same time, introducing a weight noise into the network. It has been shown that by adding noise into the weights of a network, overfitting could be reduced for improving accuracy [17]. No additional circuit is required for injecting noise into the network in this way.

Only forward propagation is performed in inference. The backward propagation component is disabled and the forward propagation component uses the unaltered layer weights to compute the classification results.

3.3 Forward propagation component

In the forward propagation component, the neurons in the input layer use ESL while the neurons in the other layers use conventional SC. The block diagrams of the two types of neurons are shown in Figs. 7 and 8.

3.3.1 Stochastic computational activation unit (SCAU)

The proposed SCAU is based on the linear finite-state machines (LFSMs) introduced in [25] [26] [27]. The SCAU can approximate both $tanh$ and the clamped ReLU by changing the configuration of the FSM to meet different learning requirements.

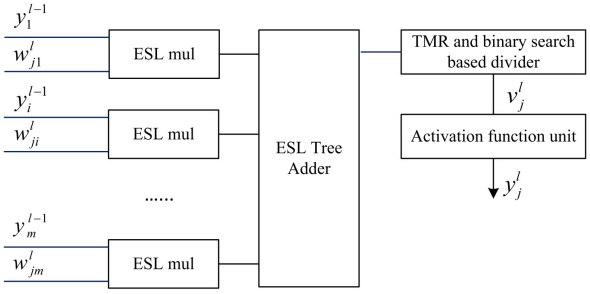


Fig. 7. The ESL based neuron j in layer l in the forward propagation. y_i^{l-1} is the input signal with dimension m, w_{ji}^l is the layer weight as in (2). ESL mul represents the ESL multiplier.

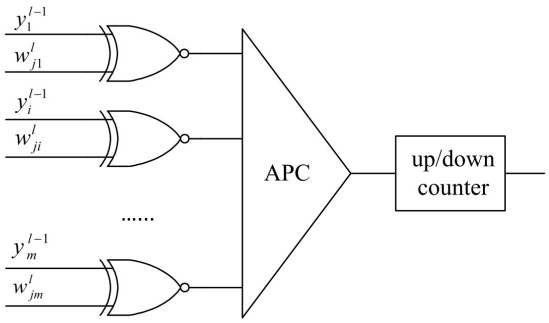


Fig. 8. The conventional SC based neuron. The circuit consists of an SC multiplier array (XNOR gates), an accumulation parallel counter (APC) and an up/down counter implementing the FSM.

A design of the stochastic $tanh$ ($Stanh$) function generator has been introduced in [5] [27] [28]. The SC absolute function has been introduced in [25]. In [12], a bounded random walking based $tanh$ ($Btanh$) circuit has been proposed as an improvement of the conventional $Stanh$ circuit. The $Btanh$ circuit consists of an accumulation parallel counter (APC) and a counter to implement the FSM for parallel input sequences. Based on these designs, an SCAU is proposed to implement different types of activations functions. The pseudo code for implementing the clamped ReLU and the $tanh$ function in SCAU is shown as following.

Code 2: Computation process of the SCAU. $not(\cdot)$ indicates the inverse operation. $SeqGen(p)$ indicates how to generate a stochastic sequence based on the given probability p. $clamp(k,\ a,\ b)$ specifies how to restrict the parameter k into the given range $[a,\ b]$.

Inputs: the sequence length m, the dimension of the input signals n, the i^{th} computation result of the APC $P_c(i)$ and the state number of the FSM in the SCAU S_{max} ($S_{max} = 2,\ 4,\ 6...$).

Outputs: updated state of FSM S and output sequence Y.

{1. State initialization}
$S_{half} = S_{max}/2$;

$S = S_{half}$;
{2. State transition and output sequence generation}
for $i = 1$ **to** m
  $\Delta S = sign(2 \times P_c(i) - n)$;
  $S = clamp(S + \Delta S,\ 1,\ S_{max})$;
  if implement the clamped ReLU function
    if $S \geq S_{half}$
      $Y(i) = not(mod(S,\ 2))$;
    else
      $Y(i) = SeqGen(0.5)$;
  else if implement the $tanh$ function
    if $S \geq S_{half}$
      $Y(i) = 1$;
    else
      $Y(i) = 0$;
end

The proposed SCAU takes the input signals in parallel by using an APC and an up/down counter. Without changing the structure of the circuit, the SCAU can implement the clamped ReLU or the $tanh$ function by reconfiguring the output of the FSM. In the SCAU, $SeqGen(\cdot)$ is used to generate a sequence for encoding the value as the lower bound of the clamped ReLU function. In the simulation, the probability encoded in this sequence is set to 0.5, which restricts the output within $[0, 1]$ in the bipolar representation.

In the SCAU, the SC multipliers compute the products of the input data and the layer weights. The APC then counts the number of '1's in the stochastic sequences generated by the SC multipliers in parallel. In this way, the APC converts the stochastic sequences into binary values, such that the SCAU correctly obtains the computation results even if the sum-of-product exceeds the range of $[-1,\ +1]$ in (2). The simulation results are shown in Fig. 9.

In the SCAU, the current state S is determined by the $sign$ function. The mod function in the algorithm is implemented by checking the least significant bit of S (Code 2). Therefore, all computations in the SCAU are implemented by shifts and comparators without using adders or multipliers.

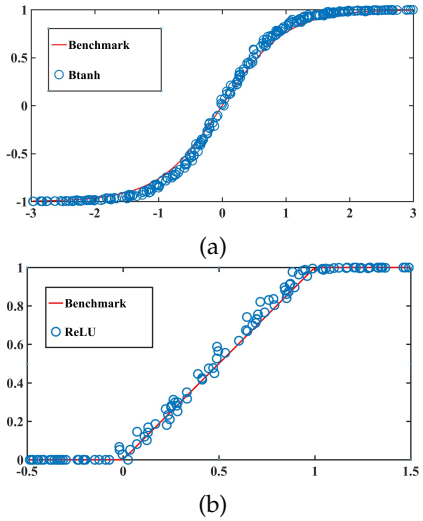


Fig. 9. Simulation result of the SCAU, with the sequence length set to 1024 bits and the maximum state number S_{max} set to 64. (a) The $tanh$ function; (b) The clamped ReLU.

3.3.2 Binary search based PE

The conversion between binary numbers and stochastic sequences significantly affects the accuracy and performance of an SC design. The conversion is usually done by a conventional PE (Fig. 4 (b)); however, the simulation results show that a conventional design requires a rather long sequence to overcome the accuracy loss when the probability of the input sequence is substantially different from that encoded in the initial value of the counter, as presented later in this section. This is mainly due to the fact that a conventional PE is based on a linear search algorithm.

To accelerate the processing speed of a PE circuit, a binary search algorithm is utilized to reduce the computational complexity from $O(2^N)$ in a linear search algorithm to $O(N)$ for a binary value of N bits. The design of the new PE is shown in Fig. 10. The circuit is divided into two parts.

Part A consists of a base register, an increment value register and one adder/subtractor module. It is designed to update and save the variable values in the binary search algorithm. The base value stored in the base register represents the currently estimated probability of the input stochastic sequence, while the increment value represents the difference between the currently estimated probability and the updated probability value.

Part B consists of three modules of DPCs, counters, comparators and a voter in a TMR structure. Its function is to compare the currently estimated probability and the observed probability of the input sequence and then decide whether to increase or decrease the base value in Part A of the circuit.

For an N bit binary number, the initial base value is 2^{N-1} and the initial increment is 2^{N-2}. At the beginning of each binary search, the base value is set for the DPCs (i.e. DPC_A, DPC_B and DPC_C) and is converted into three different stochastic sequences with the same probability. The probability of the three stochastic sequences (as the currently estimated probability) and the probability of the input sequence X are compared by the comparators COMP_A, COMP_B and COMP_C. Since the stochastic sequences are independently generated, the TMR structure reduces fluctuation errors and improves the decision accuracy, thereby reducing the required sequence length. After comparison, the three comparators vote either to increase the base value if the currently estimated probability is smaller than the observed probability, or to decrease the base value if the currently estimated probability is larger than the observed probability of the input sequence. If the two probabilities are equal, then the base value remains unchanged. Finally, the increment value is decreased by a factor of two, until it reaches 1.

The required sequence lengths for reaching the same computation accuracy are compared for the proposed PE, the conventional PE and the flip-flop based PE. The simulation results are shown in Fig. 11 for a bit width of 20 bits for the PEs. The estimation stops when the error between the PE output and the expected probability is below 1% (the initial values of the conventional PE and the proposed PE are set to 0.5). It can be seen that the required sequence length

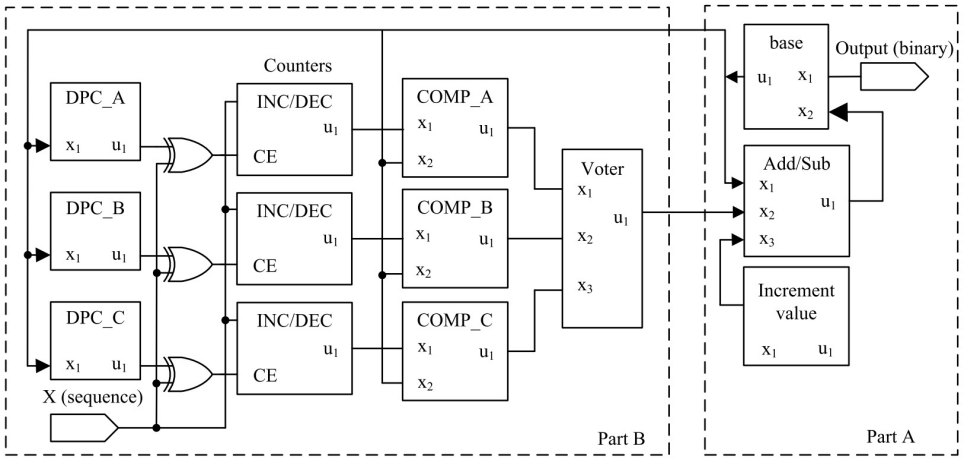


Fig. 10. Design of the TMR binary search based PE. Three counters are used to compute the probabilities encoded in the stochastic sequences generated by the three digital-probability converters (DPCs), i.e., DPC_A, DPC_B and DPC_C; COMP_A, COMP_B and COMP_C are comparators.

for the conventional PE ranges from $400,000$ bits when the probability is set to 0.5 (for the initial value of the counter in Fig. 4 (b)), to $7,200,000$ bits when the probability of the input sequence is set to 1.0 and $5,300,100$ bits when the probability is set to 0.1. These results indicate that the conventional PE design requires relatively long sequences when the probability is substantially different from the initial value encoded in the counter. As for the flip-flop based PE, it requires N_{sto} cycles to convert the probability; therefore the sequence length remains unchanged at $2^{20} = 1,048,576$ bits regardless of the probability of the input sequences.

In contrast, the sequence length required by the proposed PE is stable when the probability of the input sequences changes from 0.1 to 1.0, with the smallest value of $91,100$ bits and the largest value of $108,104$ bits. The average sequence length required by a conventional PE is $3,958,900$ bits, while the average sequence length required by the proposed PE is $96,939$ bits. Thus, the newly designed PE only requires 2.45% of the sequence length of a conventional PE and 9.24% of that of the flip-flop based PE.

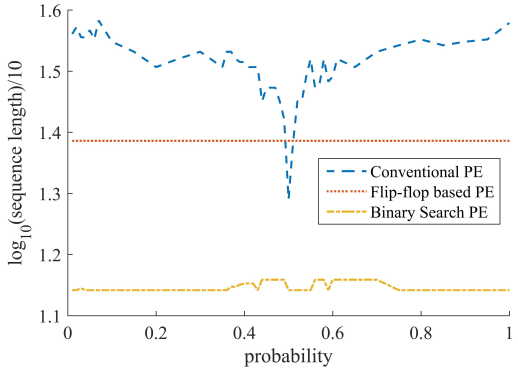


Fig. 11. Comparison of required sequence length by different PE designs vs. probability values ranging from 0.01 to 1.00.

3.3.3 *Binary search based stochastic divider*

A divider is required to convert the ESL sequences into conventional stochastic sequences as inputs to the activation circuit. Since a conventional stochastic divider relies on similar principles as the PE, the TMR and binary search based method is also applicable to the divider design, as shown in Fig. 12. For an N-bit binary number, the initial base values of the counters are 2^{N-1} and the initial increments are 2^{N-2}. The binary search continues with the increment decreasing by a factor of 2 until reaching 1. The operation of the stochastic divider is divided into two phases: a computation phase and a stabilized phase. The divider is initialized at the beginning of the computation phase and progressive precision is obtained during the computation. To evaluate the progressive precision, the mean squared error (MSE) is independently computed for each increment of 128 bits in the stochastic sequences. The simulation results of different divider implementations are shown in Fig. 13 (a).

Compared to the stepped velocity divider, the proposed design has a faster convergence speed and a higher accuracy. It shows that the TMR structure in the proposed design effectively reduces fluctuation errors and improves the decision accuracy of the binary search process. On average the proposed design requires 2.1% of the sequence length for the conventional SC divider to achieve the same accuracy. Hence, the proposed binary search based stochastic divider incurs a significantly lower latency to reach a stabilized MSE. Moreover, it also achieves a higher accuracy during the stabilized phase. Since the output value of the divider is converging to the true quotient, the values in the counters (in Fig. 12) change rather rapidly with a decreasing MSE during the computation phase. The stabilized phase starts upon the convergence of the dividers output. Due to the convergence, the counter values remain unchanged with a stable MSE during the stabilized phase.

Table 1 reports the MSEs at different phases for the three considered implementations. Only the sequences used for the stabilized phase are considered in the MSE computation. The proposed divider has the lowest MSE among the three designs at each phase configuration. The simulation results

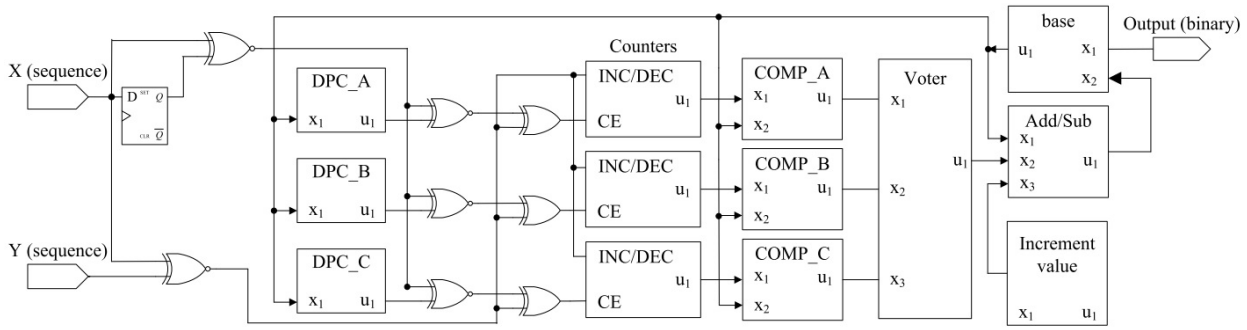


Fig. 12. Design of the TMR and binary search based divider.

suggest that the proposed divider requires at least 2048 bits for the computation phase because an insufficient sequence length in the computation phase leads to an accuracy loss during the stabilized phase. Hence, the MSE of SC dividers in the stabilized phase with the 1024-2048 phase configuration is higher than the 2048-1024 phase configuration even though the total lengths of the two sequences are the same. A longer sequence length in the stabilized phase does not guarantee a higher accuracy due to the fluctuation errors in SC. The MSE of the proposed divider reaches the smallest value when the sequence length is set to 2560 bits, i.e. 2048 bits in the computation phase and 512 bits in the stabilized phase.

Considering the classification accuracy, however, the 2048-2048 and 4096-4096 (without parallelization) are chosen as the phase configuration for the MNIST and SVHN. The MSE of the stepped velocity design is approximate $3\times$ of that of the proposed design at the same phase configuration. A conventional stochastic divider does not attain the same MSE value even by utilizing $5\times$ of the same sequence length.

TABLE 1. MSE of Stochastic Dividers in the Stabilized Phase

computation phase(bits)	Stabilized phase (bits)	Binary Search ($\times 10^{-4}$)	Stepped Velocity ($\times 10^{-4}$)	Conventional ($\times 10^{-4}$)
512	512	5.06	36.86	1755.74
1024	1024	5.22	18.37	1024.25
1024	2048	5.27	16.91	827.54
2048	512	4.09	11.59	517.84
2048	1024	4.62	13.52	534.82
2048	2048	4.84	14.51	476.84
4096	1024	4.18	12.87	269.16

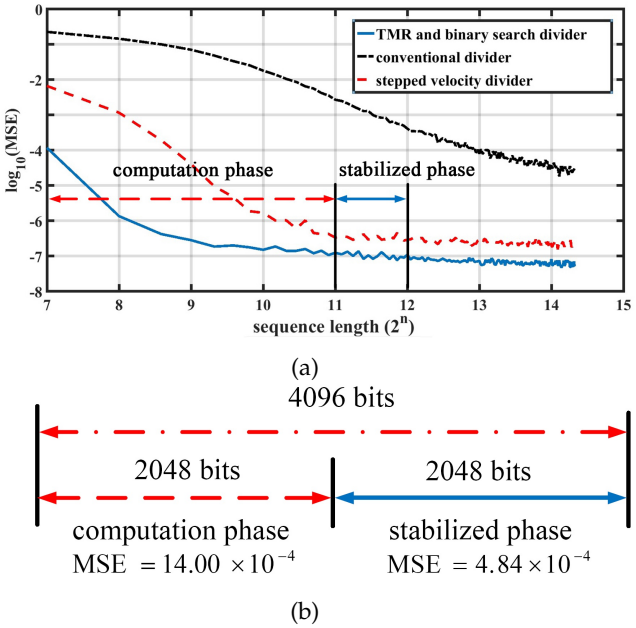


Fig. 13. (a) Convergence of the TMR and binary search based divider, conventional divider and the stepped velocity divider. The arrows indicate the 2048-2048 phase configuration of the TMR and binary search divider. (b) Mean squared error (MSE) comparison of the computation and stabilized phase with the 2048-2048 phase configuration.

The 2048-2048 phase configuration is shown in Fig. 13 (b). With this configuration, 2048 bits are required in the computation phase and additional 2048 bits are utilized in the stabilized phase. The MSE of the computation phase is 289.3% of that of the stabilized phase. Hence, to improve accuracy and energy efficiency, the sequences of 2048 bits in the computation phase are ignored and only the additional 2048 bits in the stabilized phase are used as the input signals to the activation circuit. To this end, the function of the activation circuit is suspended during the computation phase and then activated at the start of the stabilized phase. Therefore, the sequence length of the ESL divider is 4096 bits in total, but the sequence length used for the activation function is only 50% of it, i.e., 2048 bits, thereby improving the accuracy of the SC-MLP.

In summary, with the proposed binary search algorithm, the newly designed divider achieves a higher accuracy by utilizing a progressive precision with a lower latency than the stochastic designs in the literature.

3.4 Backward propagation component

In the backward propagation component, the circuit to compute the derivative of the activation functions, as in (7), and two different neurons for the gradient calculation are implemented by ESL. As per (7), when the $tanh$ function is set as the activation function, we have

$$\begin{aligned}\phi'(v_j^l(n)) &= 2(1 - tanh^2(2 \cdot v_j^l(n))),\\ &= 2(1 + y_j^l(n))(1 - y_j^l(n)).\end{aligned} \tag{15}$$

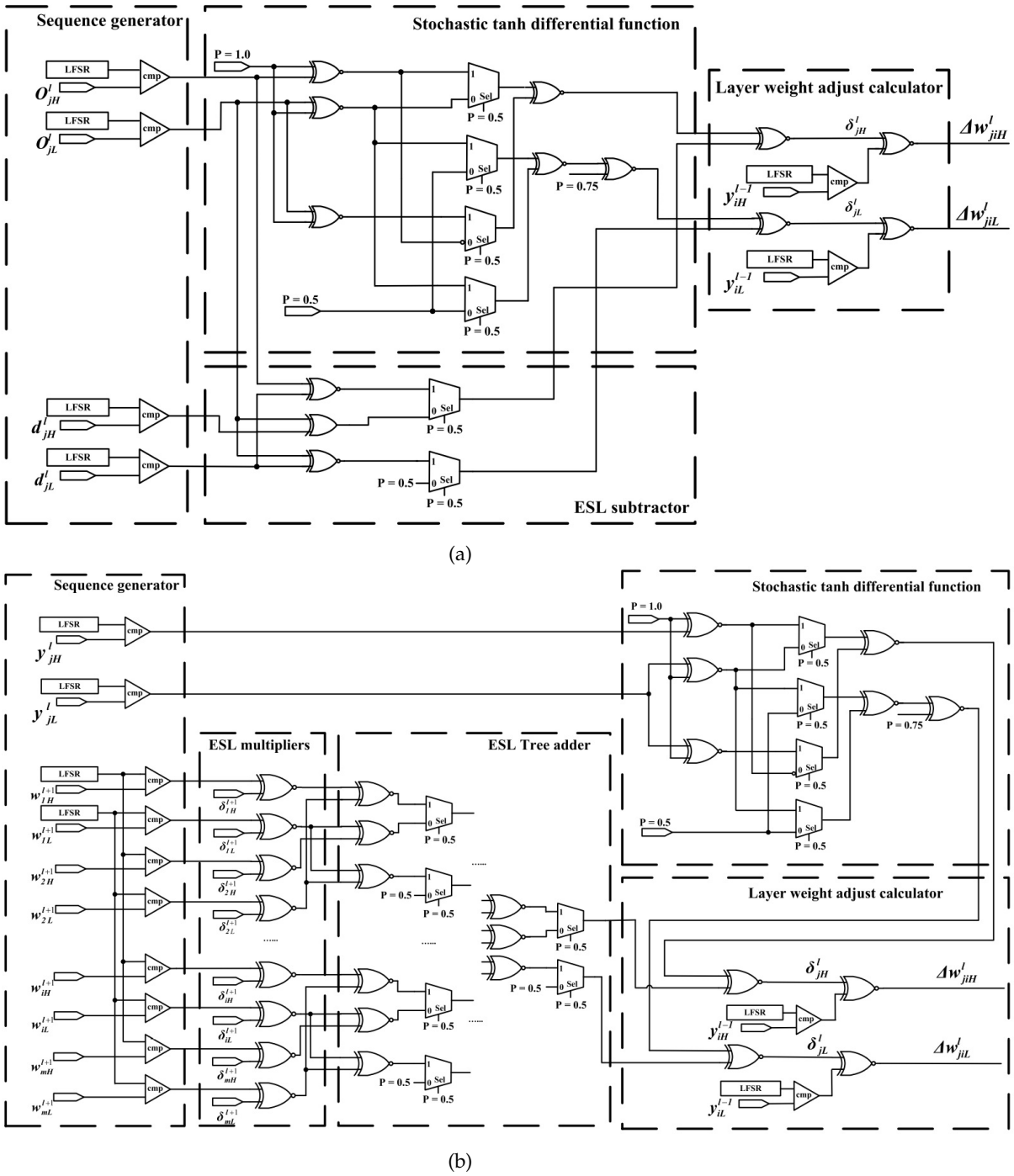


Fig. 14. The backward propagation circuits for (a) a neuron at the output layer and (b) a neuron at a hidden layer. The signals follow the same definitions in (5) (6) (7) using ESL.

It shows that the derivative of $tanh$ can be implemented by one ESL subtractor, one ESL adder and two ESL multipliers. In comparison, the derivative of the clamped ReLU function yields 0 or 1 for the current input values.

According to (7), there are two different backward propagation neurons, including neurons at output layers and neurons at hidden layers. The former implements the function $(d_j(n) - o_j^l)\phi'(v_j^l(n))$, while the latter implements the function $\phi'(v_j^l(n))\sum_i \delta_i^{l+1} w_{ij}^{l+1}(n)$. Both neurons can be implemented by using ESL subtractors, adders, multipliers and the derivative of activation functions. The designs of the two types of neurons are shown in Fig. 14. The signals in the designs are ESL sequences.

3.5 LFSR sharing structure

A structure using shared LFSRs is utilized to further reduce circuit area and power consumption. Within a neuron in the input layer, the sequences for each ESL dividend and divisor are generated from different LFSRs in the sequence generator. For the neurons in the input and hidden layers,

input sequences for each stochastic multiplier are generated from different LFSRs, so the computational accuracy is not compromised.

To reduce hardware overhead, however, the LFSRs in the DPCs for the input signals and the layer weights are all shared among different neurons in the same layer. The design using shared LFSRs is also implemented in the forward propagation circuit.

Because multipliers are one of the major components in the SC-MLP, the effect of sharing LFSRs is estimated by considering the multiplier. The area of a 32-bit floating point multiplier (FPmul) is 2791 μm^2, synthesized by the Synopsys Design Compiler in an industrial 28 nm technology library. The area of a conventional SC bipolar multiplier (SCmul, essentially an XNOR gate) is 0.8 μm^2 and the area of a 16-bit LFSR is 47 μm^2. Assume that the FPmul is combinational and the energy of the circuit linearly increases with the area, the energy consumption of the FPmul is approximately proportional to 2791.

Assume that the sequence length for the SCmul is initially 4096 bits, it is then decreased by 50% due to the use of progressive precision in the divider design. Since each SCmul requires 2 LFSRs for generating the input stochastic sequences, the energy consumption of the SCmul without sharing the LFSRs is estimated to be proportional to

$$(0.8 + 2 \times 47) \times (4096\ cycles \times 50\%) = 1.94 \times 10^5, \quad (16)$$

which is $69.6\times$ of the energy consumption of the FPmul. In the LFSR sharing structure, an LFSR is shared among all the neurons in the same layer. For a 5-layer network with the structure of 704-2048-2048-2048-10, the sharing ratio is

$$r = (704 + 2048 \times 3 + 10)/5 = 1.37 \times 10^3. \quad (17)$$

The energy consumption is estimated to be proportional to

$$(0.8 + 2 \times 47/r) \times (4096\ cycles \times 50\%) = 1778.8, \quad (18)$$

which is only 63.7% of the energy consumption of the FPmul.

This analysis indicates that when the size of the network is large, the energy consumption of the LFSR is negligible because of a large sharing ratio. This resource sharing scheme reduces considerable area and power consumption of a stochastic circuit without significantly affecting the accuracy of the computed result.

4 Experiments

Two datasets, MNIST [14] and Street View House Numbers (SVHN) [15], are used to compare the performance of the SC-MLP, a BNN, and fixed-point and floating-point MLPs with respect to accuracy, area and energy consumption.

MNIST consists of a training set with 60K samples and a testing set with 10K samples of 28×28 grayscale handwritten images labeled as '0' to '9'. In our experiment, the pixel values are scaled to $[0,\ 1]$. The neural network structure for MNIST is set to 784-200-100-10, i.e. one input layer with 784 neurons, two hidden layers with 200 and 100 neurons and one output layer with 10 neurons.

SVHN is a real-world image dataset consisting of 604K training samples and 26K testing samples of 32×32 RGB images. The dataset contains pictures of house numbers (from '0' to '9') from Google Street View images. In the experiment, the dataset is first processed by edge detection and then converted into grayscale images. Five pixels are removed from the left and right sides of each image to reduce the distraction, so the size of the images is changed to 32×22. The pixel values of the images are also scaled to $[0,\ 1]$. The neural network for SVHN is set to 704-2048-2048-2048-10 for all MLP models.

The modules of the SC-MLPs are implemented in both Matlab and VHDL. The results are compared to ensure that both implementations generate the same or very similar results. To speed up the simulation, the parallelization of the SC-MLP is set to $16\times$ with the sequence length varying from 32 to 2048 bits. A BinaryNet based BNN, a fixed point and a floating point network is implemented and compared to the proposed design with respect to different metrics such as accuracy, area and energy consumption. The bit width of the accurate layer weights in the BNN and the fixed point MLP is set to 8 and the bit width of the floating point implementation is 32.

4.1 Accuracy comparison

In each experiment, the neural network is trained by a 10-fold validation on the training datasets. The last fold of the training data is used to compute the validation error to check the early stop condition if the current validation error is at least 3.0% higher than the minimum validation error in history. At the beginning of each experiment, the datasets are randomly divided into 10 folds and the layer weights are randomly initialized. Each experiment is repeated for 10 times. The learning curve of the SC-MLP application on MNIST is shown in Fig. 15, with a $16\times$ parallelization and the sequence length set to 256 bits. The accuracy of the SC-MLP is given by the average of the testing accuracy at epoch 200 in all experiments unless an early stop occurs. No early stop has been reported with a learning rate initialized to 0.01.

Fig. 16 shows the average testing error rates of the SC-MLP (with $tanh$ as the activation function) for different sequence lengths with a $16\times$ parallelization. The classification accuracy of MNIST improves rapidly from 73.54% to 97.95% with the sequence length changing from 32 bits to 256 bits (before parallelization). However, it is not efficient to further improve the accuracy by using longer sequences. The increase from 0.02% to 0.12% in accuracy by doubling

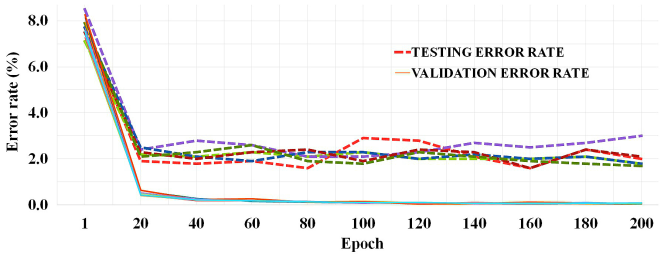


Fig. 15. Learning curve of the SC-MLP application on MNIST, with a $16\times$ parallelization and the sequence length set to 256 bits.

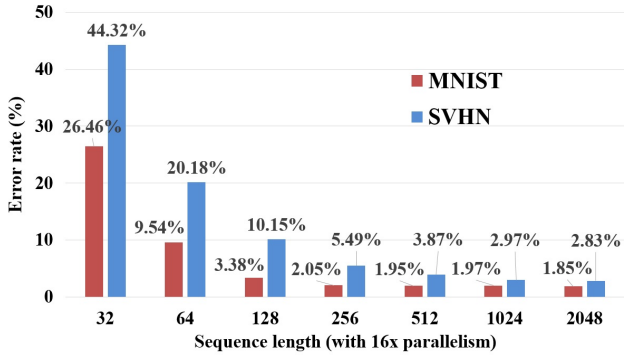


Fig. 16. Inference error rate of the SC-MLP (with $tanh$ as the activation function) with sequence length changing from 32 bits to 2048 bits.

the sequence length since 256 bits is rather modest. A similar pattern is also found for the classification accuracy of SVHN, with a sequence length varying from 32 bits to 512 bits. Therefore, 256 bits for MNIST and 512 bits for SVHN are selected for comparison with the other models. The comparison results for accuracy are listed in Table 2. The SC-MLP with the batch normalization only for the input dataset is denoted as SC-MLP-A while the implementation with the batch normalization for all layers is denoted as SC-MLP-B.

TABLE 2. Accuracy of Network Models

Model	MNIST	SVHN
SC-MLP-A ($tanh$)	97.95%	96.13%
SC-MLP-A (clamped ReLU)	97.92%	95.86%
SC-MLP-B ($tanh$)	93.67%	91.38%
SC-MLP-B (clamped ReLU)	93.32%	91.10%
BNN	97.55%	95.62%
Fixed Point NN	98.10%	96.46%
Floating Point NN	99.27%	97.47%
integral stochastic NN [10]	97.73%	-
SC NN [12]	97.59%	-

As per the simulation results, the SC-MLP-A achieves a higher accuracy on average compared to the BNN implementation and previous SC results. For MNIST, the accuracy of the SC-MLP-A is lower than the floating-point implementation by 1.32% and the fixed-point implementation by approximately 0.15%. This difference is 1.34% and 0.33% for SVHN. There is no significant difference by using $tanh$ or the clamped ReLU as the activation function in the SC-MLP. The $tanh$ function achieves a slightly higher accuracy on average, 0.03% higher for MNIST and 0.27% higher for SVHN.

The accuracy of the SC-MLP-A is $4.28\% - 4.75\%$ higher than the SC-MLP-B. This occurs because in the SC-MLP-B, the outputs of the activation functions are shifted out of the conventional SC range by the batch normalization, which causes an inaccuracy in the computation of the SC-MLP. Therefore, the batch normalization is eliminated in the hidden layers of the SC-MLP to reduce accuracy loss.

4.2 Hardware efficiency

The ASIC implementations for the different models are assessed with respect to area and energy consumption. The models are implemented in VHDL and synthesized by the Synopsys Design Compiler in ST's 28 nm technology library. The results are shown in Fig. 17.

The synthesis results indicate that the SC-MLP requires the lowest area and energy consumption for processing each data sample among the different models. The area of the SC-MLP is from $80.7\% - 87.1\%$ of the BNN, from $40.7\% - 45.5\%$ of the fixed-point implementation and from $28.5\% - 30.1\%$ of the floating-point implementation. The energy of the SC-MLP is from $71.9\% - 93.1\%$ of the BNN, from $38.0\% - 51.0\%$ of the fixed-point implementation and from $18.9\% - 23.9\%$ of the floating-point implementation.

As discussed, the batch normalization is eliminated in the hidden layers in the SC-MLP. However, in the BNN, the batch normalization is included and the layer weights are updated and then stored with full length (8 bits) at the end of the backward propagation. Therefore, the SC-MLP achieves a slightly lower area and energy consumption compared to the BNN.

It is interesting to note that the area and energy consumption of the SC-MLP for SVHN is even slightly lower than those of the floating-point implementation for MNIST. As SVHN is a more complex dataset than MNIST, it in-

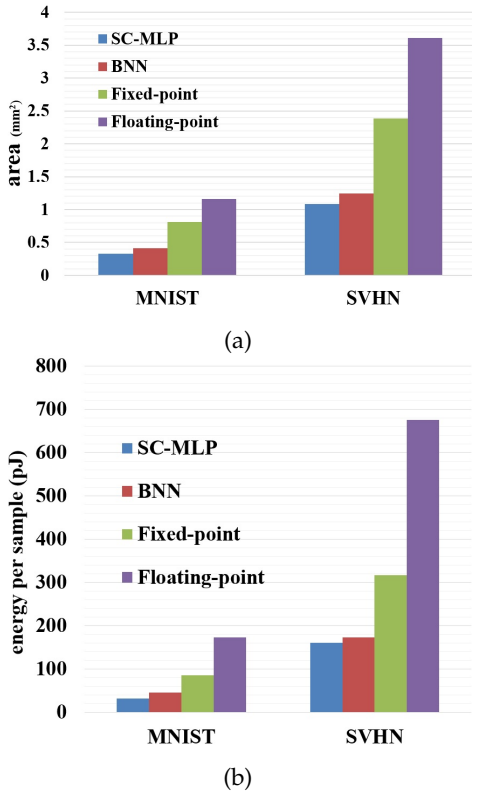


Fig. 17. (a) Area and (b) energy consumption of different network applications on MNIST and SVHN. The sequence length of SC-MLP is set to 256 bits for MNIST and 512 bits for SVHN; the bit-width of the BNN and fixed-point network is set to 8; the bit-width of the floating-point implementation is set to 32.

dicates that with similar hardware resources, the SC-MLP can potentially handle more difficult classification problems than a floating-point implementation with appropriate data pre-processing.

Table 3 shows the latency of the designs for the applications. For the MNIST application, the sequence length of the SC-MLP for processing each data sample is set to 256 bits after applying a $16\times$ parallelization. By contrast, the fixed- and floating-point implementations require 262 and 392 clock cycles for processing each sample. The latency of the SC-MLP for processing each sample is 2.27 μs when operating at the maximum frequency. It is 104.8% of that of the fixed-point design (2.17 μs) and 66.6% of that of the floating-point design (3.42 μs). For the SVHN application, the sequence length of the SC-MLP is set to 512 bits. The latency of the SC-MLP is 135.2% of that of the fixed-point implementation and 84.5% of that of the floating-point implementation. Although the computation speed is in general a challenge for SC [29], the proposed design shows no significant disadvantage in performance compared to binary designs.

TABLE 3. Latency of Network Designs

	Network	Frequency (MHz)	Cycle (/sample)	Latency (μs/sample)
MNIST	SC-MLP	112.4	256	2.27
	Fixed	120.5	262	2.17
	Floating	114.7	392	3.42
SVHN	SC-MLP	104.4	512	4.90
	Fixed	112.3	407	3.62
	Floating	108.7	630	5.80

5 Conclusion

In this paper, a stochastic computational (SC) neural network is proposed as a novel design of a multi-layer perceptron (MLP). A binary search based SC divider and a reconfigurable stochastic computational activation unit (SCAU) are proposed for the forward and backward propagation components. Using a hybrid network structure consisting of conventional SC and extended stochastic logic (ESL) circuits, the SC-MLP circuit efficiently performs the complete backward propagation algorithm.

Compared to a fully-implemented ESL network, the hybrid network structure reduces circuit area and energy consumption without loss in the classification accuracy. An LFSR sharing scheme is further utilized to improve the energy efficiency of the stochastic circuit. By using the binary search based probability estimators (PEs) and dividers, the SC-MLP requires significantly shorter sequences than conventional SC designs by achieving a progressive precision. The SCAU is reconfigurable to perform different activation functions. It can also process input sequences in parallel, thus improving the flexibility and performance of the design.

The simulation results show that the SC-MLP can solve classification problems by adjusting the network structure, implementing different activation functions and modifying the layer weights. With a similar accuracy, the proposed design achieves lower area and energy consumption compared to a binarized neural network (BNN). By incurring a slight decrease in accuracy, the SC-MLP offers considerable advantages in circuit area and energy consumption compared to floating- and fixed-point implementations with a similar performance.

References

[1] S. S. Haykin, *Neural networks and learning machines*, vol. 3. Pearson Upper Saddle River, NJ, USA, 2009.

[2] Valle, Maurizio, Caviglia, D. D, and G. M. Bisio, "An experimental analog VLSI neural network with on-chip back-propagation learning," *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 3, pp. 231–245, 1996.

[3] H. Hikawa, "A digital hardware pulse-mode neuron with piecewise linear activation function," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1028–1037, 2003.

[4] B.R.Gaines, "Stochastic computing systems," in *Advances in information systems science*, pp. 37–172, 1969.

[5] B. D. Brown and H. C. Card, "Stochastic neural computation. I. computational elements," *IEEE Transactions on computers*, vol. 50, no. 9, pp. 891–905, 2001.

[6] B. D. Brown and H. C. Card, "Stochastic neural computation. II. soft competitive learning," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 906–920, 2001.

[7] S. Liu and J. Han, "Hardware ODE solvers using stochastic circuits," in *Proceedings of the 54th Annual Design Automation Conference*, p. 81, 2017.

[8] Y. Ji, F. Ran, C. Ma, and D. J. Lilja, "A hardware implementation of a radial basis function neural network using stochastic logic," in *DATE*, pp. 880–883, 2015.

[9] 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, 2016.

[10] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[11] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou, "FPGA implementation of a deep belief network architecture for character recognition using stochastic computation," in *IEEE CISS*, pp. 1–5, 2015.

[12] 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 *DAC*, p. 124, 2016.

[13] 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," in *ASPLOS*, pp. 405–418, 2017.

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

[15] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, p. 5, 2011.

[16] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[17] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE ICASSP*, pp. 6645–6649, 2013.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456, 2015.

[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *NIPS*, pp. 4107–4115, 2016.

[20] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, pp. 3123–3131, 2015.

[21] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *ACM FPGA*, pp. 65–74, 2017.

[22] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, "Design, evaluation and fault-tolerance analysis of stochastic FIR filters," *Microelectronics Reliability*, vol. 57, pp. 111–127, 2016.
[23] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.
[24] N. Onizawa, S. Koshita, and T. Hanyu, "Scaled IIR filter based on stochastic computation," in *IEEE MWSCAS*, pp. 1–4, 2015.
[25] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1474–1486, 2014.
[26] A. Alaghi and J. P. Hayes, "On the functions realized by stochastic computing circuits," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 331–336, 2015.
[27] R. Wang, J. Han, B. F. Cockburn, and D. G. Elliott, "Stochastic circuit design and performance evaluation of vector quantization for different error measures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3169–3183, 2016.
[28] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi, "Efficient hardware implementation of the hyperbolic tangent sigmoid function," in *IEEE ISCAS*, pp. 2117–2120, 2009.
[29] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *DAC*, p. 59, 2015.