An Energy-Efficient and Noise-Tolerant Recurrent Neural Network using Stochastic Computing

Yidong Liu, Leibo Liu, Fabrizio Lombardi, *Fellow, IEEE,* and Jie Han, *Senior Member, IEEE*

***Abstract*—Recurrent neural networks (RNNs) are widely used to solve a large class of recognition problems, including prediction, machine translation, and speech recognition. The hardware implementation of RNNs is, however, challenging due to the high area and energy consumption of these networks. Recently, stochastic computing (SC) has been considered for implementing neural networks and reducing the hardware consumption. In this paper, we propose an energy-efficient and noise-tolerant long short-term memory based RNN using SC. In this SC-RNN, a hybrid structure is developed by utilizing SC designs and binary circuits to improve the hardware efficiency without significant loss of accuracy. The area and energy consumption of the proposed design are between 1.6%-2.3% and 6.5%-11.2%, respectively, of a 32-bit floating-point implementation. The SC-RNN requires significantly smaller area and lower energy consumption in most cases compared to an 8-bit fixed point implementation. The proposed design achieves a higher noise tolerance compared to binary implementations. The inference accuracy is from 10%-13% higher than a floating-point design when the noise level is high in the computation process.**

***Index Terms*—Stochastic computing, recurrent neural network, long-short term memory, energy efficient, noise tolerance.**

I. Introduction

As a type of deep neural networks (DNNs), recurrent neural networks (RNNs) are widely used for solving prediction, machine translation, and speech recognition problems [1]. The long short-term memory (LSTM) structure has been introduced to avoid catastrophic errors in the computation process, so it leads to significant accuracy improvements for RNNs [2]. Thus, it has become one of the most useful RNNs.

Recently, there have been multiple designs for improving the hardware efficiency of an LSTM-RNN. A balance aware pruning algorithm has been introduced to improve the parallel processing efficiency [3]. The Fast Fourier Transform (FFT) and inverse FFT have also been utilized to reduce the complexity of matrix multiplication in the LSTM [4]. A structured compression technique has been utilized to compress the weight matrices [5]. However, it still remains a challenge to implement an LSTM-RNN on resource-limited systems, such as a mobile device or an embedded system due to the high computational complexity, area cost, and power consumption.

Recent advances in stochastic computing (SC) provide such an opportunity to improve the hardware efficiency of NNs by reducing the hardware of fundamental arithmetic circuits such as adders, subtractors and multipliers [6] [7] [8]. Nonlinear functions can be approximated by Bernstein polynomials and Maclaurin series [9]. SC designs have been proposed to implement radial basis function-based neural networks [10], multilayer perceptrons (MLPs) [11] [12], convolutional neural networks (CNNs) [13] [14] and deep belief networks (DBNs) [15] [16]. The applicability of SC to RNNs has also been briefly explored in [17]. In such applications, noise tolerance is of ultimate importance.

In this paper, an energy-efficient LSTM-RNN is proposed by leveraging the hardware efficiency of SC circuits. A hybrid structure utilizing SC and binary circuits are designed to improve the hardware efficiency and retain the accuracy in inference. The LSTM memory block is implemented by binary circuits and approximate parallel counter based SC circuits. To reduce the memory requirement, internal stochastic sequences are converted into binary values for storage. Three datasets are used for the evaluation of the RNN design: the Reder grammar [18], Japanese vowels [19] and TIMIT [20]. Simulation results show that the proposed SC-RNN requires a significantly smaller area, lower energy consumption in most cases, and at the same time, achieves a comparable accuracy and higher noise tolerance compared to conventional binary implementations.

The remainder of this paper is organized as follows. Section II introduces the background for RNNs, LSTM, and stochastic logic. Section III presents the proposed design. Section IV shows the application and simulation results. Section V concludes the paper.

II. Background

A. Recurrent Neural Networks (RNNs)

The structure of a typical fully connected RNN (FCRNN) is shown in Fig. 1 [21]. The FCRNN consists of two layers: a concatenated input-feedback layer (C-layer) and a processing layer (P-layer). If the longest delay of an input is set to p and each delayed external signal is assigned to a neuron, p neurons results in total. Assume that the P-layer consists of N neurons, the inputs of each neuron in the P-layer consist of N dimensional feedback signals $y_q(t-1), q = 1, 2, ..., N$, p dimensional delayed external signals $s(t-j), j = 1, 2, ..., p$ and a bias. For the n^{th} neuron in the P-layer, the layer

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

Y. Liu, and J. Han are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 1H9 Canada e-mail: {yidong1, jhan8}@ualberta.ca.

L. Liu is with the Institute of Microelectronics Department, Tsinghua University, HaiDian District, Beijing 100084, China e-mail: liulb@tsinghua.edu.cn

F. Lombardi is with Department of Electrical and Computer Engineering, Northeastern University, Boston 02115, MA USA (e-mail: lombardi@ece.neu.edu).

weights form a $(N + p + 1)$ dimensional weight vector $W_n = [w_{n,1}, w_{n,2}, ..., w_{n,N+p+1}]$.

The real-time recurrent learning (RTRL) algorithm is one of the most widely used learning algorithms for RNNs [21]. In the RTRL, the inference process is similar to that in a conventional multilayer perceptron. For the n^{th} neuron, the output is given by

$$y_n(t) = \Phi(v_n(t)), n = 1, 2, ..., N \tag{1}$$

and

$$v_n(t) = \sum_{l=1}^{N+p+1} w_{n,l}(t) \cdot u_l(t). \tag{2}$$

where $\Phi(\cdot)$ is the activation function and $u_l(t) \in \{y_1(t-1), y_2(t-1), ..., y_N(t-1), 1, s(t-1), s(t-2), ..., s(t-p)\}, l = 1,\ 2,\ ...,\ N+p+1$ [21].

B. Long-short Term Memory (LSTM)

The LSTM is one of the most widely-used learning algorithms for RNNs. Different from the FCRNN, it utilizes gate units to constrain the feedback signals, so the gradient does not either quickly reduce or increase during the feedback process [2], so the entire process is rather stable. It is reported that the LSTM achieves higher performance compared to conventional RNN learning algorithms [2] [22].

In the LSTM-RNN, the neurons are implemented by memory blocks. A single-cell memory block stores the current internal state and includes three types of gate units, an input gate (in), an output gate (out) and a forget gate (φ) (Fig. 2). In this work, we follow the algorithm introduced in [2] and [22]. The gate units implement the same activation function as

$$f(x) = \frac{1}{1+e^{-x}},\ f(x) \in [0, 1]. \tag{3}$$

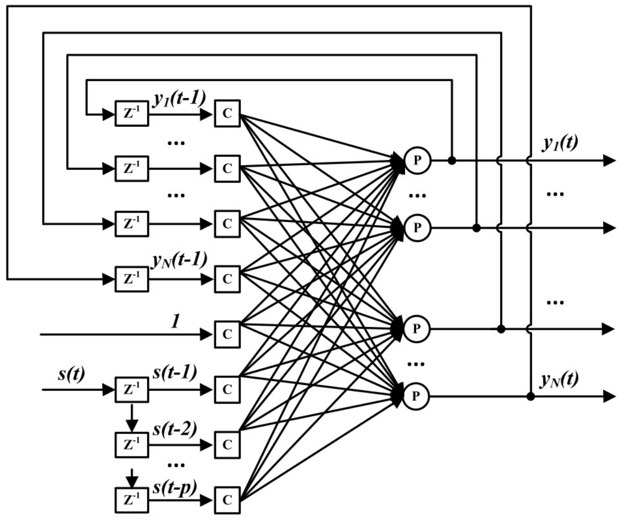


Fig. 1: Structure of a fully connected RNN, with the C nodes denoting neurons in the concatenated input-feedback layer (C-layer) and the P nodes denoting neurons in the processing layer (P-layer).

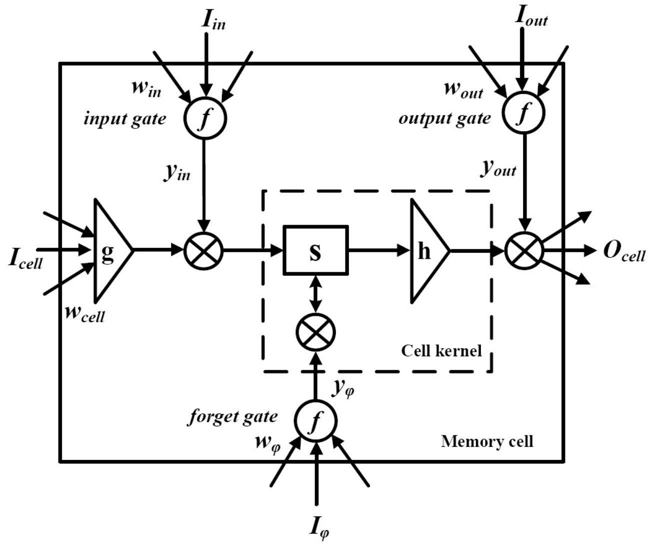


Fig. 2: Functions of a memory cell for LSTM-RNNs [22]. f is the activation function implementing the gates defined by (3). g and h are activation functions defined by (8) and (10). I_l ($l \in \{cell, in, out, \varphi\}$) are the input signals of the cell and gates. y_{in}, y_{out} and y_φ are the output signals generated by the gate units. O_{cell} is the output signal of the cell. s represents the internal state of the cell. The cell kernel is introduced in Section 3.

Assume that k is the index of memory blocks, v is the index of cells in the k^{th} block, c_k^v is the v^{th} cell in the k^{th} block. $w_{p,q}$ is the layer weight between unit p and unit q. $I_l(t)$ ($l \in \{out, in, \varphi\}$) represents all the input signals connected to the gates at time t, including the outputs from other cells and the feedback signals from the gates in the cell. The gating signals of the input, output and forget gates at time t are given by

$$y_l(t) = f(net_l(t)), l \in \{out, in, \varphi\}, \tag{4}$$

where

$$net_l(t) = \sum w_l \cdot I_l(t), l \in \{out, in, \varphi\}. \tag{5}$$

The input signals of the cell are defined as $I_{cell}(t-1)$ and the layer weights are defined as w_{cell}, so

$$net_{c_k^v}(t) = \sum w_{cell} \cdot I_{cell}(t). \tag{6}$$

The internal state of the cell $S_{c_k^v}$ is determined by the input signals of the cell, the input gate, the forget gate and the previous internal states:

$$S_{c_k^v}(t) = \begin{cases} 0 & t = 0, \\ y_{\varphi_k}(t) \cdot S_{c_k^v}(t-1) + y_{in_k}(t) \cdot g(net_{c_k^v}(t)) & t > 0, \end{cases} \tag{7}$$

where $g(\cdot)$ is defined as

$$g(x) = \frac{4}{1+e^{-x}} - 2 = 2 \cdot tanh(\frac{x}{2}),\ g(x) \in [-2, 2]. \tag{8}$$

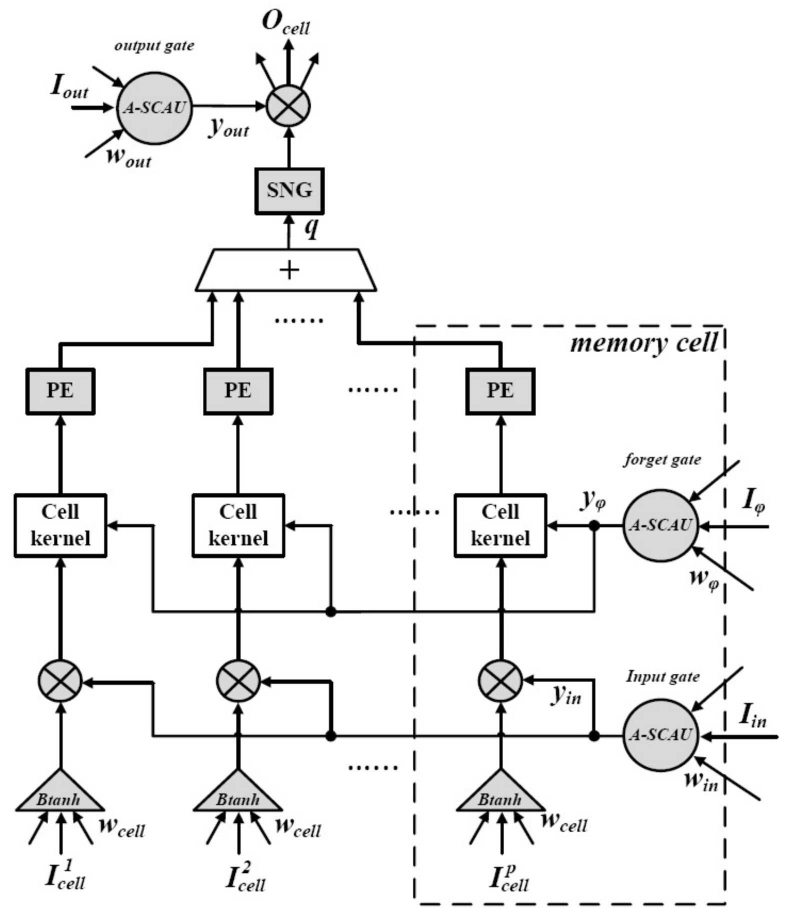


Fig. 3: Structure of a multi-cell memory block in the SC RNN. The multipliers are implemented by SC circuits. The A-SCAU implements the gate units. The $Btanh$ circuit implements the activation function g defined by (8). The cell kernel updates the internal state and computes the output of a cell. PE is the probability estimator that converts stochastic sequences into binary values. SNG is the stochastic number generator. The circuits in grey are implemented by or for SC. The cell kernel is implemented by both SC and binary circuits.

The output signal of the cell O_{cell} in the hidden layer is computed through the memory cell to the output gate. It is defined as $O_{c_k^v}(t)$ for cell c_k^v at time t, and computed as

$$O_{c_k^v}(t) = y_{out_j}(t) \cdot h(S_{c_k^v}(t)), \tag{9}$$

where $h(\cdot)$ is the activation function defined as

$$h(x) = \tanh(\frac{x}{2}),\ h(x) \in [-1, 1]. \tag{10}$$

In Fig. 2, the function h, the internal state of the cell s and the multiplication are considered as the cell kernel for the convenience of hardware implementation. The design of the cell kernel is introduced in Section 3.

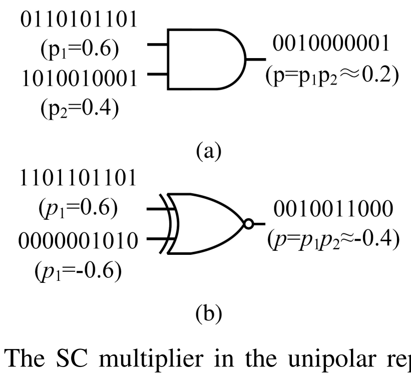


Fig. 4: (a). The SC multiplier in the unipolar representation. (b) The SC multiplier in the bipolar representation [7].

C. Stochastic Computing

In stochastic computing (SC), assume that there are p 1's in a q-bit random binary bit stream, the stream then encodes the value p/q in the unipolar representation, and the value $(2p - q)/q$ in the bipolar representation [7] [23]. The bit stream is generated by a stochastic number generator (SNG), consisting of a random number generator (RNG) and a comparator [24] (Fig. 5). The SC encoding significantly reduces the complexity of an arithmetic circuit compared to binary designs. For example, a bipolar multiplier is implemented by an XNOR gate and a weighted adder is implemented by a multiplexer, as shown in Fig. 4. The $tanh$ function can be implemented by a finite state machine (FSM) with the state transition diagram shown in Fig. 6 [7]. However, the computation range of SC is limited in $[0, 1]$ in the unipolar

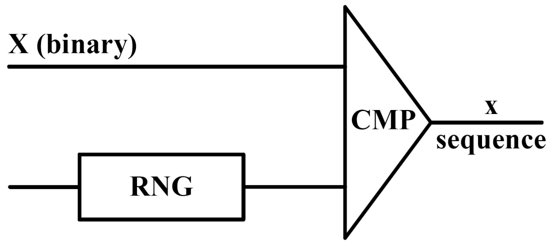


Fig. 5: Design of an SNG, consisting of an RNG and a comparator [24].

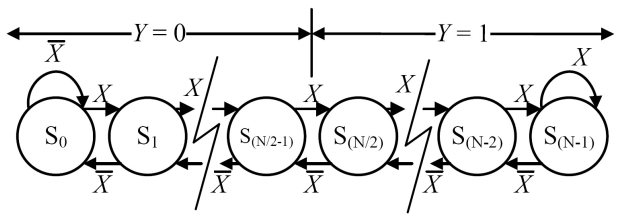


Fig. 6: The state transition diagram of the FSM-based $tanh$ circuit, implementing $y = tanh(\frac{N}{2}x)$. X indicates that the input bit is '1' and $\bar{X}$ indicates that the input bit is '0'. Y is the output bit [7].

representation and $[-1, +1]$ in the bipolar representation.

III. SC-RNN DESIGN

A. Overall Design

In the proposed design, the computational components are implemented by a hybrid structure consisting of SC and binary circuits. The inference data and the network structure (weights) are stored in external memory elements. The training is performed by utilizing Matlab functions and the layer weights are saved. The inference process is implemented by SC and binary circuits with the computed layer weights.

B. Hybrid Structure of the Memory Block

The SC-RNN design is based on a multi-cell memory block [22]. The output of the memory block is defined as the sum of the output of each memory cell in the block. The structure of a multi-cell memory block is shown in Fig. 3. The block includes p cells. The outputs of the input gates (y_{in}) and forget gates (y_{φ}) are shared among the cells in the same memory block, as shown in Fig. 3. The input signals of different cells ($I^k_{cell}, k \in 1,\ 2,\ ...,\ p$) are separated. The gate units are implemented by the approximate SC activation unit (A-SCAU) [16]. The activation function g in (8) is implemented by the $Btanh$ circuit [25]. The values of the signals of the gates and the layer weights are encoded by stochastic sequences in the bipolar representation. Note that the probabilities encoded in the sequences for the input vectors (I_{in}, I_{out}, I_{φ}, and I^k_{cell}, $k \in 1,\ 2,\ ...,\ p$) are the same as per the LSTM definition [22]. However, the sequences are generated separately to reduce the correlation.

The cell kernel is utilized to update the value of the internal state $S_{c^v_k}$ in (7) and compute the output signal of the cell following (9) and (10). The probability estimator (PE) is utilized to convert the stochastic sequences into binary values [12]. In the memory cell, the PE, the A-SCAU, the $Btanh$ function and the multiplication are implemented by SC circuits while the cell kernel is designed using binary and SC methods. The variable q in Fig. 3 is computed using binary adders. The register storing the value of q is also used as the memory element of the intermediate computation result [26]. Thus, the system stores intermediate computation results in the binary format to reduce the required memory overhead instead of storing the stochastic sequences. The binary results are then re-converted to stochastic sequences by SNGs. The output signals of the memory block are used as inputs to the other memory blocks, thus interacting with the internal states of the cells in the other memory blocks.

1) Gate units: As per (3), the gate units (including the input, output and forget gates) can be implemented by an SC circuit for the sigmoid activation function. The gate units are implemented by the approximate SC activation unit (A-SCAU) in [16] (Fig. 3). The A-SCAU is utilized to implement multiple types of activation functions and reduce at the same time the hardware overhead. The design of the A-SCAU is shown in Fig. 7. The linear approximation unit (LAU) implements multiple activation functions with the generalized form of

$$\psi(x) = min(1, max(p, \frac{1}{r}x + s)), \tag{11}$$

where p, r and s are configurable parameters. The sigmoid function (3) can be approximated by setting the configuration to $\{p = 0,\ r = 4,\ s = 1/2\}$; the ReLU function can be implemented by setting the configuration to $\{p = 0,\ r = 1,\ s = 0\}$. It is shown in [16] that the output of the LAU is determined by the sum of the probabilities of the input stochastic sequences. As a result, the circuit is immune to correlations between the input sequences. Therefore, the RNGs for generating the input sequences can be shared among different neurons with no loss in computation accuracy. Because the RNGs are one of the most costly units in SC circuits [24], the sharing strategy significantly reduces the hardware cost.

2) Cell kernel: The cell kernel in Fig. 3 updates the internal state and the output signal of the cell. The SNG and multiplication are implemented for SC. The FSM is utilized to implement SC $Btanh$ function and generates the SC sequence

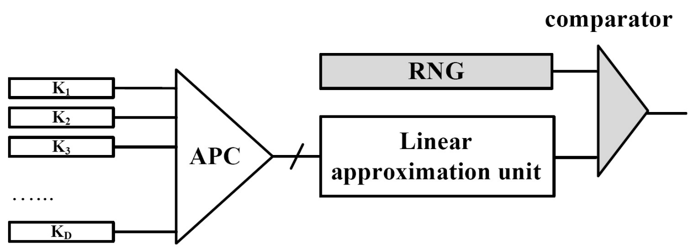


Fig. 7: Design of the A-SCAU [16], including an accumulative parallel counter (APC), a linear approximation unit (LAU), an RNG and a comparator. $K_i, i = 1,\ 2,\ ...,\ D$, is the input stochastic sequence. The circuits in grey are implemented by or for SC.

as the output signal. The accumulative parallel counter (APC) and state processing unit (SPU) are based on binary circuits. According to (7), when $t \neq 0$, there exists

$$\begin{aligned} S_{c_k^v}(t) &= y_{\varphi_k}(t)\cdot S_{c_k^v}(t-1) + y_{in_k}(t)\cdot g(net_{c_k^v}(t)) \\ &= y_{\varphi_k}(t)\cdot S_{c_k^v}(t-1) + y_{in_k}(t)\cdot 2\cdot tanh(\frac{net_{c_k^v}(t)}{2}) \\ &= y_{\varphi_k}(t)\cdot S_{c_k^v}(t-1) + y_{in_k}(t)\cdot tanh(\frac{net_{c_k^v}(t)}{2}) \\ &\quad + y_{in_k}(t)\cdot tanh(\frac{net_{c_k^v}(t)}{2}). \end{aligned} \tag{12}$$

It shows that the internal state $S_{c_k^v}(t)$ can be computed by SC circuits with the value of each addend in (12) restricted within $[-1,+1]$. The design for the cell kernel is shown in Fig. 8.

In Fig. 8, to update $S_{c_k^v}(t)$, the signal y encodes the value of $y_{\varphi_k}(t)$ and is multiplied with the signal encoding the value of $S_{c_k^v}(t-1)$ by the SC multiplier, resulting in a signal x encoding the value of $y_{\varphi_k}(t)\cdot S_{c_k^v}(t-1)$ in the bipolar representation. The signals z and z' encodes the same value of $y_{in_k}(t)\cdot tanh(\frac{net_{c_k^v}(t)}{2})$ in the bipolar representation but they are generated independently. All the signals are set as inputs of the APC.

The SPU is designed to compute the value of $S_{c_k^v}(t)$ at each step of the updating process. It is implemented by binary circuits. The design of the SPU is explained as follows. According to (12), the value of $S_{c_k^v}(t)$ is computed by the sum of the values encoded by the signal x, z, and z' in Fig. 8 in the bipolar representation. Assume that the number of the input signals is D (here, $D = 3$) and that each input signal is parallelized to α folds, which means that each input signal is generated by α synchronized SNGs with the same probability. Assume that the number of 1's in the j^{th} sequence for the i^{th} signal is $Q_{i,j}$ after parallelization, the number of 1's in the input sequences is computed by the APC as $\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j}$.

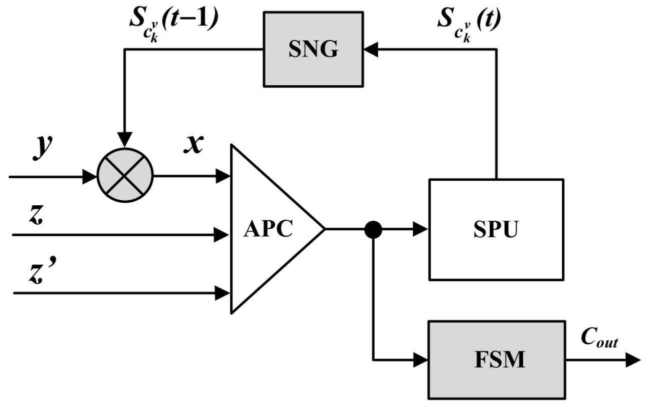


Fig. 8: Design of the cell kernel, including an SC multiplier, an APC, an SNG, an FSM for implementing the SC $Btanh$ function and an SPU for updating the internal state. C_{out} is the output sequence, encoding the value of $h(S_{c_k^v}(t))$. The circuits in grey are implemented by or for SC. The internal design of the SPU is shown in Fig. 10.

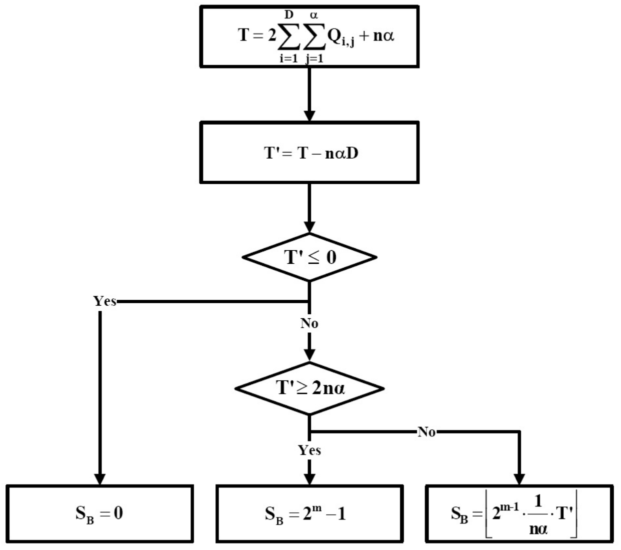

$T = 2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} + n\alpha$
$T' = T - n\alpha D$
$T' \le 0$
Yes
No
$T' \ge 2n\alpha$
No
Yes
$S_B = 0$
$S_B = 2^m - 1$
$S_B = \lfloor 2^{m-1}\cdot\frac{1}{n\alpha}\cdot T' \rfloor$

Fig. 9: The algorithmic flowchart for the SPU. T, T': temporary variables used to store the intermediate results in the computation. The definitions of other signals are the same as those in (17) and (18).

For n-bit sequences and due to the bipolar representation, the value of $S_{c_k^v}$ computed by the APC, S, is expected to be

$$S = \frac{1}{\alpha}\sum_{i=1}^{D}\sum_{j=1}^{\alpha}(2\frac{Q_{i,j}}{n} - 1). \tag{13}$$

Note that the range of S is $[-D,+D]$. In the SPU design, the value of S is considered to be clamped into $[-1,+1]$ for the bipolar representation in SC. The clamped value, S_c, is given by

$$S_c = \begin{cases} -1, & S \leqslant -1, \\ +1, & S \geqslant +1, \\ S, & others, \end{cases} \tag{14}$$

However, according to (9), the output of the network is not affected because it is only determined by the activation function $h(S_{c_k^v}(t))$, which produces a similar output value for the clamped input. Taking (13) into (14) gives us

$$S_c = \begin{cases} -1, & 2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} + n\alpha \leqslant n\alpha D, \\ +1, & 2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} - n\alpha D \geqslant n\alpha. \\ S, & others, \end{cases} \tag{15}$$

Let $T = 2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} + n\alpha$ and $T' = T - n\alpha D$, by scaling the value of $\frac{1}{2}(S_c+1)$ into an m-bit binary vector, the output of the SPU, S_B, is obtained as

$$S_B = \begin{cases} 0, & T \leqslant n\alpha D, \\ 2^m - 1, & T' \geqslant 2n\alpha, \\ {S_B}', & others, \end{cases} \tag{16}$$

where ${S_B}'$ is given by

$$\begin{aligned} {S_B}' &= \lfloor (2^m - 1)\cdot \tfrac{S+1}{2} \rfloor \\ &= \left\lfloor \tfrac{(2^m-1)}{2n\alpha}(2\sum_{i=1}^{D}\sum_{j=1}^{\alpha} Q_{i,j} + n\alpha(1-D)) \right\rfloor. \end{aligned} \tag{17}$$

S_B is the integer approximation of $S_{c_k^v}(t)$ in Fig. 8 and $S_B{}'$ is approximated by

$$S_B{}' = \left\lfloor (2^m - 1) \cdot \frac{1}{2n\alpha} \cdot T' \right\rfloor \approx \left\lfloor 2^{m-1} \cdot \frac{1}{n\alpha} \cdot T' \right\rfloor. \quad (18)$$

The algorithmic flowchart of the SPU is shown in Fig. 9. When n and α are set to values in a power of 2, the multipliers and dividers in the SPU can be implemented by shift registers. The computation circuit of T and T' is implemented by an accumulator, an adder, a subtractor and shift registers. The circuit design is shown in Fig. 10.

The output of the cell (C_{out} in Fig. 8) is implemented by the $Btanh$ circuit [25], which consists of an APC and an FSM in Fig. 8. The output signal encodes the value of $h(S_{c_k^v}(t))$ in the bipolar representation. The FSM is implemented by an up-down counter following the algorithm in [25]. The APC is reused between the SPU and the $Btanh$ circuit, thus reducing the hardware cost.

The APC can be replaced by an approximate parallel counter using a similar design method of [27] to further reduce the hardware cost. Assume that there are 9 input sequences for the APC, the design of the approximate parallel counter is shown in Fig. 11.

Table I shows the detailed area breakdown of the cell kernel, including three parallelization configurations: 1-fold (no parallelization), 4-fold, and 8-fold parallelization. It means that each input signal is generated by 1, 4, and 8 synchronized SNGs with the same probability. By an 8-fold parallelization, for example, a 256-bit sequence is implemented by eight 32-bit sequences (generated by independent SNGs) in parallel. The areas of the FSM, the accumulator in the SPU, and the control unit of the cell kernel are listed separately while the areas of the other parts of the SPU are divided into combinational and sequential circuits. The second column in the table represents the percentage of the area of each component in the cell kernel. Note that the control unit of the cell kernel requires 27.7%

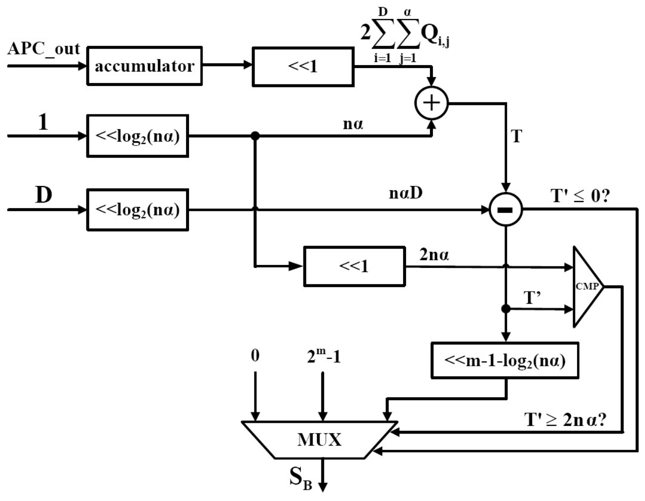


Fig. 10: The circuit design of the SPU. CMP: comparator. $<<$: left-shift register. APC_out is the output signal of the APC. The definitions of the signals are the same as those in (17) and (18).

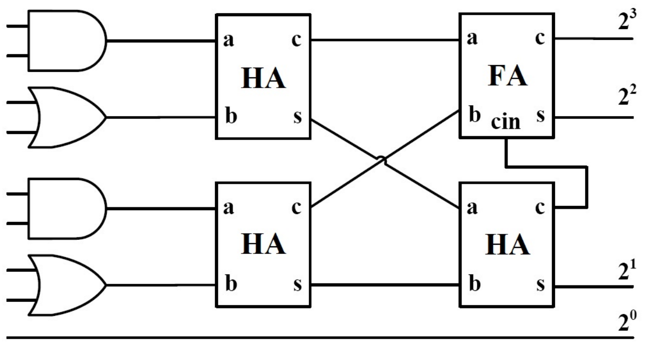


Fig. 11: Design of the approximate parallel counter with 9 input sequences. HA represents the half adder and FA represents the full adder.

of the total area when there is no parallelization due to the complex operations in the cell kernel and the SPU. This drops to 19.0% and 12.8% with 4-fold and 8-fold parallelization. As the core of the LSTM, the cell kernel can be adjusted and utilized to implement different types of LSTM models such as the convolutional LSTM network [28], combined with the SC components in CNNs [13].

TABLE I: Area breakdown of the cell kernel (μm^2)

Paral.	FSM (%)	Accum. (%)	Contr. (%)	Combi. (%)	Seque. (%)	total
1	214.3 (19.7%)	392.9 (36.1%)	301.6 (27.7%)	59.5 (5.5%)	119.1 (11.0%)	1087.4
4	357.2 (17.9%)	654.9 (33.1%)	377.0 (19.0%)	198.4 (10.0%)	396.9 (20.0%)	1984.4
8	535.8 (18.1%)	851.3 (28.8%)	377.0 (12.8%)	396.9 (13.4)	793.8 (26.9%)	2954.8

IV. Experiments

The SC-RNN is utilized for the prediction of symbol generation using the Reder grammar dataset [18], and speech recognition using the Japanese vowels [19] and TIMIT datasets [20]. The performances of the SC-RNN and binary LSTM RNNs are assessed with respect to accuracy, area and energy consumption.

A. Reder Grammar Problems

The symbol generating rules for the Reder grammar is shown in Fig. 12. The Reder grammar sequentially generates symbol strings from the left node by following the edges and appending the associated symbols to the current string until the right-most node is reached. Edges are randomly chosen by the probability of 50%. In inference, the networks read strings, one symbol at a time to predict the next symbol.

Similarly to [22], 256 strings are randomly generated with an average length of 16 as the testing dataset. The training of the RNN is implemented in binary circuits by 10-fold validation, with each fold including 10^5 randomly generated strings as training datasets. The LSTM is set into two different structures, which, respectively, consist of 3 and 4 memory blocks with 2 and 1 memory cells within each block. The

TABLE II: Inference accuracy comparison for Reder Grammar dataset

Method	Structure (block, cell)	Length (bits)	Acc (%) (no noise)	Acc (%) (10 dB noise)
SC RNN	(3, 2)	32	21	0
		64	73	30
		128	100	85
	(4, 1)	32	20	0
		64	73	25
		128	100	83
32-bit FP	(3, 2), (4, 1)	n/a	100	60
8-bit FxP	(3, 2), (4, 1)	n/a	100	60

inference circuits are implemented by 32-bit floating-point (FP), 8-bit fixed point (FxP) and SC designs. Noises are added in the computation process as soft errors (or flipping faults) with the signal-to-noise ratio (SNR) computed by

$$SNR\ (dB) = 10 \cdot log_{10}(\frac{P_s}{P_n}), \qquad (19)$$

where P_s is the power of the signal and P_n is the power of noise. The inference accuracy is shown in Table II.

With no noise in the computation process, both the 8-bit FxP design and the 32-bit FP design achieve 100% in inference accuracy for the (3, 2) and (4, 1) structures. For the SC design, a longer sequence length leads to a higher precision, thus improving the computation accuracy. Therefore, the inference accuracy of the SC-RNN increases with sequence length, from 21% to 100% when the sequence length varies from 32 to 128 bits. With 128 bits, the SC RNN achieves the same accuracy as any of the binary implementations in both considered structures. However, with an SNR of 10 dB in the computation process, the inference accuracy of the SC-RNN is 85% whereas the accuracy of the binary designs is reduced to 60%. This result indicates that the SC-RNN achieves a higher noise tolerance.

B. Voice Recognition: Japanese Vowels

The Japanese vowels dataset includes 9 male speakers uttering the Japanese vowel /ae/ successively. Each instance consists of 12 features (cepstral coefficients) with the length of 14 to 26 in time series. The training dataset includes 270 instances (30 utterances by 9 speakers) and the testing dataset includes 370 instances ($24-88$ utterances by the same 9 speakers). In inference, each testing sample is classified to the correct speaker, resulting in 9 labels in total.

The network has a 12-120-9 structure, i.e., with 12 neurons in the input layer, 120 LSTM neurons in the hidden layer and 9 neurons in the output layer. The training process is performed by utilizing Matlab functions and the inference process is performed in SC, 8-bit FxP, and 32-bit FP circuits, respectively, for comparison. The sequence length of the SC implementation for the Japanese vowel dataset is 256 bits prior to parallelization.

In inference, Gaussian white noise is added to the datasets and the computation process for evaluating the noise tolerance of different implementations. The inference accuracy is shown in Fig. 13. The simulation results with no noise are plotted at an SNR of 20 dB for easier illustration and readability.

Two cases are considered in the experiment. For SC1 and Binary1 in the first case, the Gaussian white noise is added in the computation process, but no noise in the dataset. For the same SNR, the inference accuracy of the SC-RNN is higher than that of the FP implementation, except for the noise-free result. With no noise, the SC-RNN achieves a lower accuracy (93.8%) than the FP design (94.9%). For an SNR of 5 dB, the accuracy of the FP design is 73.4%, whereas the accuracy of the SC-RNN is 86.6%.

For SC2 and Binary2 in the second case, a 5 dB noise is added to the dataset, in addition to the noise in the computation process. The average accuracy of Binary2 is higher than that of Binary1, indicating that the noise in the training dataset improves the inference accuracy in the FP implementation due to regularization [29] [30] [31]. The inference accuracy for SC2 ($89.3-93.8\%$) is slightly higher than that for SC1. The inference accuracy for SC2 exceeds that for Binary2 when the SNR is lower than 12 dB. In both cases, the noise in the computation process significantly affects the accuracy of the FP circuit; therefore, the SC design achieves a higher noise tolerance than the binary designs.

C. Voice Recognition: TIMIT

The TIMIT dataset has been designed for the development and evaluation of automatic speech recognition systems.

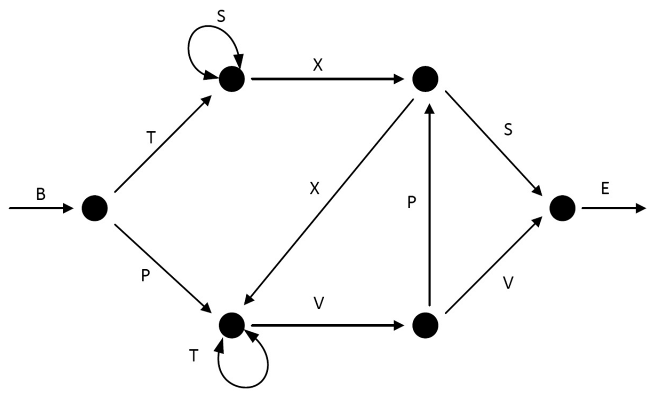


Fig. 12: The symbol generating rules for the Reder grammar.

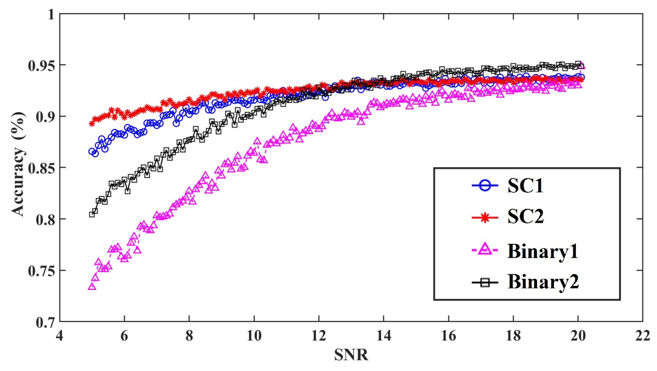


Fig. 13: The Inference accuracy of networks for the Japanese vowels dataset with noise at different SNRs.

TABLE III: Hardware efficiency for the Reder grammar networks

	SC-RNN (4, 1)	SC-RNN (3, 2)	FP RNN (4, 1)	FP RNN (3, 2)	FxP RNN (4, 1)	FxP RNN (3, 2)
Area (μm^2)	513.0	568.0	7380.1	8662.7	1844.1	2011.1
Power (mW)	0.025	0.031	0.419	0.491	0.085	0.093
Latency (μs/sample)	1.34	1.45	1.12	1.17	0.47	0.53
Energy (nJ/sample)	0.034	0.045	0.47	0.57	0.039	0.049

TABLE IV: Hardware efficiency for the Japanese vowel and TIMIT networks

	Area ($\times 10^6\ \mu m^2$)				Energy (μJ)				Latency (μs)			
Structure	SC (4-fold)	SC (8-fold)	FP	FxP	SC (4-fold)	SC (8-fold)	FP	FxP	SC (4-fold)	SC (8-fold)	FP	FxP
12-120-9	0.35	0.56	16.6	4.68	0.03	0.03	0.25	0.02	0.51	0.33	0.46	0.15
12-(250, 250)-48	1.46	2.32	65.9	17.8	0.12	0.13	1.23	0.12	1.03	0.65	0.58	0.17
12-(500, 500)-48	3.52	5.80	212.8	59.6	0.46	0.48	7.08	0.61	1.24	0.75	0.75	0.20
12-(250, 250)-(250, 250)-48	2.93	4.64	124.7	37.4	0.31	0.34	3.48	0.28	1.67	1.05	0.86	0.25

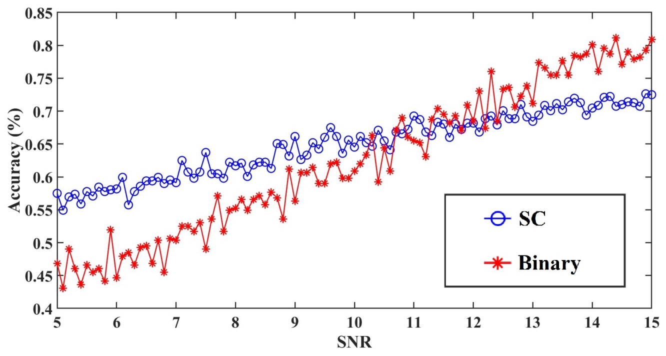


Fig. 14: The inference accuracy of networks for the TIMIT dataset with noise at different SNRs.

TIMIT contains a total of 6300 sentences, 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States.

The training process of the network is performed by utilizing Matlab functions. For inference, both the SC and the binary circuits (32-bit FP and 8-bit FxP) are implemented for this network. The sequence length of the SC-RNN is given by 256 bits prior to parallelization. The structure of the network is given by 12-(250, 250)-(250, 250)-48, that is, the network consists of one 12-neuron input layer, one 48-neuron output layer and 2 hidden layers with 250 memory blocks and each memory block including 8 cells.

Gaussian white noise is added to the computation process, with the SNR varying from 5 dB to 15 dB. The results are shown in Fig. 14. The simulation results with no noise are plotted at an SNR of 15 dB for easier illustration and readability. With no noise, the highest accuracy of the FP implementation is 81.9%, similar to those in [3] [4] [32] [33] while the accuracy of the SC design is 10% lower. However, the SC design achieves a higher accuracy when the SNR is lower than 12 dB. For an SNR of 5 dB, the accuracy of the SC-RNN is 57%, which is more than 10% higher than the FP implementation with the same SNR. The simulation results show that the SC design achieves a significantly higher noise tolerance than the conventional FP implementation.

D. Hardware Efficiency

The ASIC design of the SC-RNNs and binary Reder grammar networks are assessed with respect to area and energy consumption for inference. All designs use VHDL models synthesized by the Synopsys Design Compiler in ST's 28 nm technology library. The power consumption of the circuit is measured by the synthesis tool. The computation time is computed by the working frequency and computation cycles and the energy is computed based on the power and computation time. For the networks solving the Reder grammar prediction problem, the sequence length of the SC-RNN is given by 128 bits, with no accuracy loss compared to the binary designs. The results are shown in Table III. The latency and energy consumption are obtained for processing one sample through the network.

The synthesis results indicate that the proposed design requires lower area and energy consumption compared to the 32-bit FP circuits. The (4, 1) network achieves lower area and energy consumption than the (3, 2) network. The area, power, and energy of the SC-RNN are respectively $6.6\% - 7.0\%$, $6.0\% - 6.3\%$ and $7.2\% - 7.9\%$ of the FP design for different network configurations. The computation latency of the SC-RNN is similar to the FP circuit due to the parallelization. The area of the proposed design is $27.9\% - 28.2\%$ of the FxP design, with a longer computation latency (ranging between $2.7\times - 2.9\times$). Overall, the energy of the SC-RNN is slightly lower at $87\% - 92\%$ of the 8-bit FxP design.

The hardware requirement of the networks for the Japanese vowel and TIMIT dataset is also found for inference and is shown in Table IV. The sequence length of the SC-RNN is given by 256 bits prior to parallelization. The structure of the Japanese vowel is set as 12-120-9. The structures of the TIMIT network are set as 12-(250, 250)-48, 12-(500, 500)-48, and 12-

(250, 250)-(250, 250)-48. The SC networks are implemented with 4-fold and 8-fold parallelization, represented by SC (4-fold) and SC (8-fold) in Table IV. The SC RNNs operate at 200 MHz while the operating frequency of the FP and FxP circuit is 100 MHz. The binary circuits are not pipelined.

The latency and energy consumption in Table IV are obtained on average for processing one sample through one neuron. The synthesis results indicate that with 4-fold parallelization, the SC-RNN incurs a lower area ($1.6\% - 2.3\%$) and smaller energy consumption ($6.5\% - 11.2\%$) than the FP circuit for different network structures. These figures of merits are $5.9\% - 8.2\%$ and $75.4\% - 128.5\%$ of that of the 8-bit FxP designs. The latency of the proposed design is $2\times$ of the FP implementation and $5\times$ of the FxP implementation due to the long stochastic sequences. The proposed design achieves significantly lower power consumption, but at a lower number of frames per second due to the slower computation speed, compared to the designs of [5] [33]. However, the latency of the SC design can be further reduced by using a higher level of parallelization. For example, with an 8-fold parallelization in the SC network, the energy consumption is almost identical but the computation speed of the SC RNN is improved by $35.0\% - 39.5\%$ with an increase in area, compared to that of the SC network with a 4-fold parallelization.

V. Conclusion

In this paper, an SC design is proposed to reduce the area and energy consumption of RNNs. The circuits are implemented by SC for inference. The proposed design requires significantly smaller area and lower energy consumption in most cases at a higher latency compared to conventional binary implementations. The SC-RNN shows significant advantages in noise tolerance by achieving higher accuracy than binary designs when noise is present in the computation process.

References

[1] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *ISCA*, 2014.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the 2017 ACM/SIGDA International Symposium on FPGA*. ACM, 2017, pp. 75–84.

[4] Z. Li, S. Wang, C. Ding, Q. Qiu, Y. Wang, and Y. Liang, "Efficient recurrent neural networks using structured matrices in FPGAs," *arXiv preprint arXiv:1803.07661*, 2018.

[5] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proceedings of the 2018 ACM/SIGDA International Symposium on FPGAs*. ACM, 2018, pp. 11–20.

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

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

[8] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *DAC*, 2015, p. 59.

[9] M. H. Najafi, P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "A reconfigurable architecture with sequential logic-based stochastic computing," *ACM JETC*, vol. 13, no. 4, p. 57, 2017.

[10] 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*, 2015, pp. 880–883.

[11] J. L. Rosselló, V. Canals, and A. Morro, "Probabilistic-based neural network implementation," in *IEEE IJCNN*, 2012, pp. 1–7.

[12] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A stochastic computational multi-layer perceptron with backward propagation," *IEEE TC*, 2018.

[13] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *IEEE ICCD*, 2016, pp. 678–681.

[14] H. Sim, S. Kenzhegulov, and J. Lee, "Dps: dynamic precision scaling for stochastic computing-based deep neural networks," in *DAC*. ACM, 2018, p. 13.

[15] 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*, 2015, pp. 1–5.

[16] Y. Liu, Y. Wang, F. Lombardi, and J. Han, "An energy-efficient online-learning stochastic computational deep belief network," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 454–465, 2018.

[17] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, "Sign-magnitude SC: getting 10X accuracy for free in stochastic computing for deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[18] A. W. Smith and D. Zipser, "Learning sequential structure with the real-time recurrent learning algorithm," *International Journal of Neural Systems*, vol. 1, no. 02, pp. 125–131, 1989.

[19] M. Kudo, J. Toyama, and M. Shimbo, "Multidimensional curve classification using passing-through regions," *Pattern Recognition Letters*, vol. 20, no. 11-13, pp. 1103–1111, 1999.

[20] J. S. Garofolo, "TIMIT acoustic phonetic continuous speech corpus," *Linguistic Data Consortium*, 1993.

[21] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[22] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[23] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *DATE*, 2014, p. 76.

[24] ——, "Survey of stochastic computing," *ACM TECS*, vol. 12, no. 2s, p. 92, 2013.

[25] 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*, 2016, p. 124.

[26] H. Sim, D. Nguyen, J. Lee, and K. Choi, "Scalable stochastic-computing accelerator for convolutional neural networks," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 696–701.

[27] 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 2017*, pp. 405–418.

[28] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015, pp. 802–810.

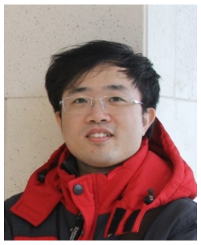
[29] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems*, 2011, pp. 2348–2356.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

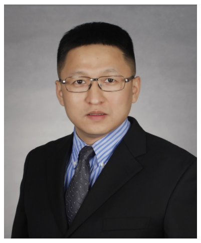
[31] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, 2013, pp. 1058–1066.

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

[33] Z. Li, C. Ding, S. Wang, W. Wen, Y. Zhuo, C. Liu, Q. Qiu, W. Xu, X. Lin, X. Qian *et al.*, "E-RNN: Design optimization for efficient recurrent neural networks in FPGAs," *arXiv preprint arXiv:1812.07106*, 2018.



Yidong Liu received the B.Eng. degree in electronic engineering and the M.Eng. degree in electronics and communication engineering from Tsinghua University, in 2007 and 2010, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Alberta, under the supervision of Dr. Jie Han. His current research interests include stochastic computing and neural networks.



Leibo Liu received the B.S. degree in electronic engineering and the Ph.D. degree with the Institute of Microelectronics, both from Tsinghua University, Beijing, China, in 1999 and 2004, respectively. He is currently a Full Professor with the Institute of Microelectronics, Tsinghua University. His current research interests include reconfigurable computing, mobile computing, and very large-scale integration digital signal processing.

Fabrizio Lombardi received the BSc. degree (Hons.) in electronic engineering from the University of Essex, U.K., in 1977, the Diploma in microwave engineering, in 1978, the masters degree in microwaves and modern optics from the University College London, in 1978, and the Ph.D. degree from the University of London, in 1982. In 1977, he joined the Microwave Research Unit, University College London. He currently holds the International Test Conference (ITC) Endowed Chair Professorship with Northeastern University, Boston. His research interests include bio-inspired and nano manufacturing/computing, VLSI design, testing, and fault/defect tolerance of digital systems. He has extensively published in these areas and coauthored/edited seven books. He was the Editor-in-Chief of the IEEE Transactions on Computers in 2007-2010 and the Inaugural Editor-in-Chief of the IEEE Transactions on Emerging Topics in Computing in 2013-2017. He is currently the Editor-in-Chief of the IEEE Transactions on Nanotechnology.

Jie Han received the B.Sc. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree from the Delft University of Technology, The Netherlands, in 2004.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His research interests include approximate computing, stochastic computation, reliability and fault tolerance, nanoelectronic circuits and systems, novel computational models for nanoscale and biological applications.

Dr. Han and his coauthors received the Best Paper Award at the International Symposium on Nanoscale Architectures 2015 (NanoArch 2015) and Best Paper Nominations at the 25th Great Lakes Symposium on VLSI 2015 (GLSVLSI 2015), NanoArch 2016 and the 19th International Symposium on Quality Electronic Design (ISQED 2018). He was nominated for the 2006 Christiaan Huygens Prize of Science by the Royal Dutch Academy of Science. His work was recognized by *Science*, for developing a theory of fault-tolerant nanocircuits in 2005. He served as the General Chair for GLSVLSI 2017 and the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2013), and the Technical Program Committee Chair for GLSVLSI 2016 and DFT 2012. He is currently an Associate Editor of the IEEE Transactions on Emerging Topics in Computing (TETC), the IEEE Transactions on Nanotechnology and Microelectronics Reliability.