An Energy-Efficient Bayesian Neural Network Implementation Using Stochastic Computing Method

Xiaotao Jia, *Member, IEEE*, Huiyi Gu, Yuhao Liu, Jianlei Yang, *Member, IEEE*, Xueyan Wang, *Member, IEEE*, Weitao Pan, *Member, IEEE*, Youguang Zhang, *Member, IEEE*, Sorin Cotofana, *Fellow, IEEE*, and Weisheng Zhao, *Fellow, IEEE*

***Abstract*— The robustness of Bayesian neural networks (BNNs) to real-world uncertainties and incompleteness has led to their application in some safety-critical fields. However, evaluating uncertainty during BNN inference requires repeated sampling and feed-forward computing, making them challenging to deploy in low-power or embedded devices. This article proposes the use of stochastic computing (SC) to optimize the hardware performance of BNN inference in terms of energy consumption and hardware utilization. The proposed approach adopts bitstream to represent Gaussian random number and applies it in the inference phase. This allows for the omission of complex transformation computations in the central limit theorem-based Gaussian random number generating (CLT-based GRNG) method and the simplification of multipliers as AND operations. Furthermore, an asynchronous parallel pipeline calculation technique is proposed in computing block to enhance operation speed. Compared with conventional binary radix-based BNN, SC-based BNN (StocBNN) realized by FPGA with 128-bit bitstream consumes much less energy consumption and hardware resources with less than 0.1% accuracy decrease when dealing with MNIST/Fashion-MNIST datasets.**

***Index Terms*— Bayesian neural network (BNN), energy efficiency, Gaussian random number generator, stochastic computing (SC).**

Manuscript received 30 November 2022; revised 19 February 2023; accepted 27 March 2023. Date of publication 3 May 2023; date of current version 4 September 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62006011, Grant U20A20204, Grant 62072019, and Grant 62004011; and in part by the 111 Talent Program under Grant B16001. *(Xiaotao Jia and Huiyi Gu contributed equally to this work.) (Corresponding authors: Jianlei Yang; Weisheng Zhao.)*

Xiaotao Jia is with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing 100191, China, and also with the Beihang Hangzhou Innovation Institute Yuhang, Hangzhou 310023, China.

Huiyi Gu, Yuhao Liu, and Youguang Zhang are with the School of Electronic and Information Engineering, Beihang University, Beijing 100191, China.

Xueyan Wang and Weisheng Zhao are with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing 100191, China (e-mail: weisheng.zhao@buaa.edu.cn).

Jianlei Yang is with the BDBC, School of Computer Science and Engineering, Beihang University, Beijing 100191, China (e-mail: jianlei@buaa.edu.cn).

Weitao Pan is with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China.

Sorin Cotofana is with the Computer Engineering Laboratory, Delft University of Technology, 2628 CD Delft, The Netherlands.

Digital Object Identifier 10.1109/TNNLS.2023.3265533

I. Introduction

DEEP neural networks (DNNs) show extensive application prospect in artificial intelligence (AI). With several promising neural network models and techniques [1], DNNs have been promoted to many fields to enhance intelligence, including object classification [2], natural language processing [3], medical analysis [4], and autonomous driving [5]. DNNs are making significant impact on the world's social activity and economy and becoming the top candidate for real-world applications. It can even exceed human performance in some of these fields [6], [7]. Despite the tremendous success of DNNs, there are still some disadvantages in some aspects. In the training phase, DNNs normally use maximum likelihood estimation (MLE) to construct the loss function and use optimization algorithms, such as stochastic gradient descent, to obtain the optimal parameter value. This training method is effective, but it is susceptible to overfitting, which makes researchers spending much effort [8]. One of the important reasons is that the use of MLE ignores any uncertainty in the proper weight values. Moreover, other disadvantages of DNNs also affect their performance, such as the lack of theoretical backbone [9], data hungry, gradient vanish [10], and easy to be fooled [11].

Bayesian neural networks (BNNs) or Bayesian deep learning (BDL) (Fig. 1) combines the Bayesian method with neural network and demonstrates promising prospects in addressing these shortcomings. Training and inference phase in BNNs are formulated with neural network models based on the probability theory. The Bayesian method takes prior knowledge into consideration to deal with real-world uncertainty and incomplete information. It is an effective supplement to the existing learning methods of DNNs. With the support of Bayesian mathematical theory and DNN nonlinear fitting models, BNNs are inherently robust to address the overfitting issue [12]. In general, the prior and posterior distributions of parameters are modeled by Gaussian distribution. Several probability programming languages (PPLs) have been developed based on widely used frameworks, such as Tensorflow-based Edward [13], Tensorflow Probability [14], ZhuSuan [15], and Pytorch-based Pyro [16]. These PPLs make learning BNNs' posterior

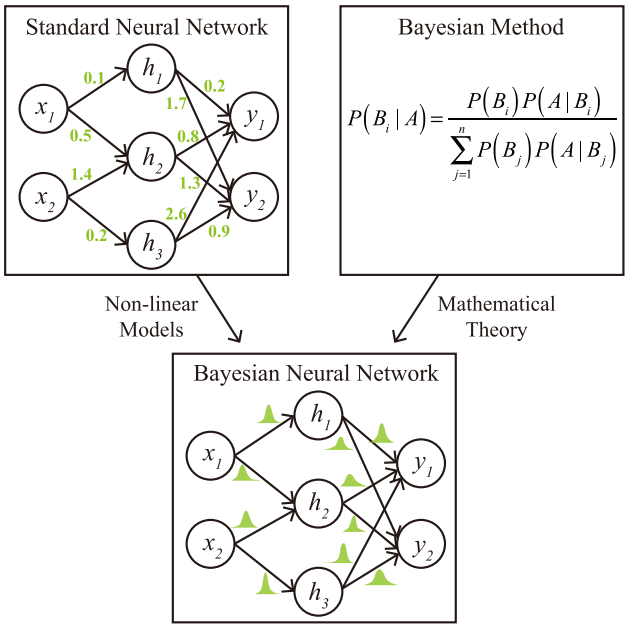

Fig. 1. Relationship among standard neural network, Bayesian method, and BNN.

distribution easier with different training datasets. BNNs are becoming increasingly prevalent in some safety-critical applications [17], [18].

Different from DNNs, BNNs need to sample synaptic weights based on well-trained posterior distributions, and each weight needs to be sampled several times. In fact, high computation complexity and high-power consumption have become two main factors that restrict the development of BNNs. We observe that random number sampling process is the crucial step in both training and inference phase and accounts for a large proportion in terms of inference latency, energy consumption, and hardware utilization. Usually, in inference phase, T neural network instances are sampled based on BNNs' posterior distribution so as to model the inputs' uncertainty. Generally speaking, T is larger than 10, or even 100 in some cases. The massive sampling and feed-forwarding operations make BNNs computationally intractable. Even though BNNs incur higher energy consumption and latency when compared with standard neural networks, the outperforming of BNNs in terms of small-data training, uncertainty estimation, and other aspects makes it worthy to be studied.

Several novel strategies have been proposed to accelerator BNN inference from different perspectives, such as sampling optimization [19], [20], [21], [22], [23], dataflow optimization [24], [25], [26], FPGA implementation innovate [19], [22], [23], [26], [27], [28], and computing-in-memory architecture [20], [25], [29]. Cai et al. [19] explore the design space for massive amount of Gaussian variable sampling tasks in BNNs and propose an FPGA-based hardware accelerator design. An efficient BNN inference approach is proposed in [24] to reduce redundant computations based on feature decomposition and memory strategy. Wu et al. [25] propose to exploit the intrinsic stochastic behaviors of analog resistive random access memory (RRAM) to generate the required distribution of BNN. Yang et al. [20] use spintronic devices to design spin-based GRNG to improve the overall hardware performance. A resource-efficient weight sampling method is proposed by [21] using inversion transform sampling and a lookup table (LUT)-based function approximation for hardware implementation. In [22], quadratic nonlinear activation functions are employed to free the sampling process.

Wan et al. [26] propose an accelerating design that intelligently skips the redundant computations of dropout masks and zero-corresponding computations. Dorrance et al. [29] from Intel Lab leverage a C-2C SRAM-based analog compute-in memory (CiM) macro for the multiply accumulation (MAC) operations to accelerate BNN. An FPGA-based design in [27] and [28] accelerates BNNs inferred through Monte Carlo dropout (MCD) and supports both 2-D and 3-D Bayes CNNs. Awano et al. [23] replace costly Gaussian random number generators (RNG) with Bernoulli RNG.

While these works propose efficient optimization approaches that significantly improve the efficiency of BNN, certain issues remain that cannot be ignored. First, the optimization of BNN sampling and feedforward propagation are often treated as independent of each other, and most works only focus on one of them. This approach can result in suboptimal performance, as these two processes are closely related. Second, the existing optimization of GRNG is based on the CLT, which involves complex transformation computing that consumes a significant amount of computing resources. The CLT method requires division and square root operations, while probability multiplications are floating computations that have high computing resource requirements. Addressing these issues will be critical for further improving the acceleration of BNNs. In this article, we introduce the stochastic computing (SC) method to BNN inference phase named StocBNN. In StocBNN, both inputs and weights are represented in bitstream format to directly participate in the inference phase. Multiplication could be simplified as AND operation. In this manner, complex transformation in CLT-based GRNG can be omitted, and feed-forwarding propagation becomes simpler than traditional one. This work is proposed for Gaussian distribution characterized BNNs and mainly focuses on the inference phase of vision classification problem. The main contributions of this work are listed as follows.

1) An energy-efficient FPGA implementation is proposed for BNN inference phase using the SC method. Both GRNG and feed-forward stage are performed in SC domain.
2) A simplified SC domain GRNG is proposed using linear feedback shift register (LFSR) with theoretical proof. In LFSR-based GRNG, the transformation computing in the CLT method is omitted, and Gaussian random numbers are represented in bitstream format. FPGA implementation of LFSR-based GRNG shows that it can generate high-quality random numbers with less hardware cost.
3) An asynchronous parallel pipeline calculation technique is proposed to speed up the feed-forward calculation and reduce the hardware cost. It effectively increases the computing speed by about 3× with less fan-out.
4) The hardware efficiency of StocBNN is evaluated by an FPGA with 128-bit bitstream. Compared with

conventional binary radix-based BNN, StocBNN consumes much less energy consumption and hardware resources with less than 0.1% accuracy decrease when dealing with MNIST/Fashion-MNIST datasets.

The remainder of this article is organized as follows. Section II discusses some preliminaries. Section III describes in detail why and how the SC method can be introduced into BNNs. The architecture of StocBNN is introduced in Section IV. Experimental results are illustrated in Section V. Section VI gives the conclusion.

II. Preliminaries

This work focuses on performing BNN inference in SC domain to achieve better performance. SC provides a new form of data expression, which can be naturally combined with BNNs. Before the description of technical details, some preliminaries and related works are discussed in this section.

A. Bayesian Neural Networks

BNNs refer to the augmentation of standard neural networks with posterior inference to create a deep learning framework, which is able to cope with parameter uncertainty. It is a kind of neural network whose parameters are trained based on the Bayesian method. The Bayesian method is a statistical approach used to estimate certain properties of statistics. Its mathematical foundation is Bayesian theory, which can be written as follows:

$$P(W|D) = \frac{P(D|W)P(W)}{P(D)}. \tag{1}$$

Here, W means neural network parameters and D means training data of neural network. We define $P(W)$ as the prior probability distribution, which is the artificially estimated data. $P(D|W)/P(D)$ is the likelihood, and it can be regarded as an adjustment factor to make the estimated probability distribution closer to the true one. $P(W|D)$ is posterior probability distribution. By assuming prior probability of parameters and continuously learning posterior probability distribution, network parameters can be trained.

As described in [19], BNNs provide a different training method and weight/bias representation format rather than a novel neural network structure. There are three main differences between standard neural networks and BNNs. The first one is the training method. A gradient descend-based method is widely used in standard neural networks training phase, such as batch gradient descent and stochastic gradient descent, while variational inference and Markov chain Monte Carlo method tend to be faster and easier to implement in BNN training stage. The second one is parameter representation. Standard neural networks have numerical parameter values, but BNNs parameter values are defined by a probability distribution, such as Gaussian distribution and Laplace distribution. Parameters mostly are characterized by mean and variance. Variance represents a measure of uncertainty. Therefore, BNNs show outstanding performance in handling uncertain and incomplete information. Inference manner is the last difference. Standard neural networks perform the feed-forward procedure only once, while BNNs perform multiple times.

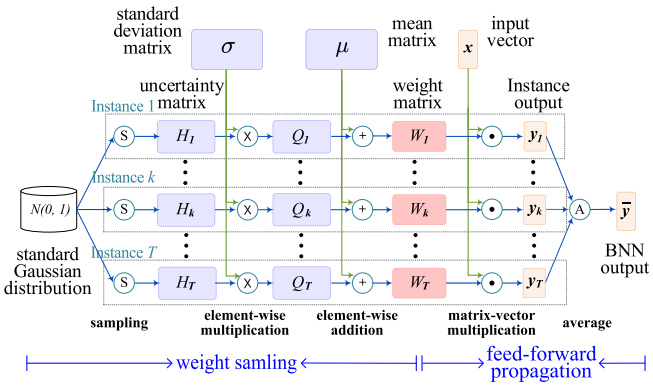

Fig. 2. Standard single-layer BNN inference dataflow [24].

B. Gaussian Random Number Generator

As the first and key step in BNN inference process, Gaussian random variables sampling requires a properly optimized GRNG. Previous works on such issue consist of both software and hardware implementations [30]. The generation method of random numbers and the quality of obtained data affect not only the difficulty of subsequent calculation but also the accuracy of final result. Here, a brief introduction of Gaussian random variable sampling is given. Suppose that ω fits Gaussian distribution whose mean value is μ and standard deviation is σ, i.e., $\omega \sim N(\mu, \sigma^2)$. Before sampling one weight w from the given distribution $N(\mu, \sigma^2)$, another random number u should be sampled first from standard Gaussian distribution [i.e., $u \sim N(0, 1)$]. Then, based on the scale-location transformation, the expected random number could be calculated as $w = u\sigma + \mu$. By randomly generating massive u, different ω values can be obtained. The standard GRNG algorithms can be classified into four distinctive categories: inversion methods, transformation methods, rejection methods, and recursion methods [30]. Among them, the CLT-based transformation method is the most widely used one, especially in hardware implementation.

C. Dataflow in BNN Inference Phase

With a well-trained BNN, the inference operation could be divided into two stages: weight sampling and feed-forward propagation. Fig. 2 shows the standard inference dataflow of a single-layer BNN. In this example, μ and σ are the well-trained distribution parameter matrices of BNN. $\boldsymbol{x}$ is the input vector.

In weight sampling stage, T concrete neural networks are instantiated through large amount of sampling operations based on weights' posterior distribution. Specifically, T uncertain matrices $H_1, H_2, \ldots, H_T$ are sampled from $N(0, 1)$, and then, according to the parameter matrices, T weight matrices $W_1, W_2, \ldots, W_T$ are transformed from T uncertain matrices using scale-location transformation. Every element in the T weight matrices fits Gaussian distribution $N(\mu, \sigma^2)$. In feed-forward propagation stage, input vector $\boldsymbol{x}$ is fed into all these T weight matrices to finish the feed-forward procedure and achieve a convincing result. T outputs $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}$ are averaged to get the final output $\bar{\boldsymbol{y}}$. Jia et al. point out that there are large amount of redundant computation in the standard dataflow, as shown in Fig. 2, and propose a

feature decomposition and memorization strategy to reform the BNN inference dataflow in a reduced manner [24]. With the proposed dataflow, the computation of a single-layer BNN could be reduced by half without any influence on inference accuracy but at the expense of 14% area overhead. In this article, the efficient inference dataflow proposed in [24] is adopted and improved.

D. Stochastic Computing

SC is a nonconventional computing paradigm, which is first proposed by von Neumann in the 1950s [31]. In SC, data are presented in the format of 0-1 sequence, which is named stochastic number (SN). The value of SN is evaluated based on the count of 1's in bitstream, and all bits are independent of each other in bitstream. With different encoding format, the value is different. In unipolar format, the value of one SN is calculated by the ratio of 1's [i.e., $P(x = 1)$] and ranges from 0 to 1. In bipolar format, the value is calculated by $2P(x = 1) - 1$ and ranges from -1 to 1. For instance, both 01001001 and 11000001 contain three ones in an 8-bit bitstream. In unipolar format, the value is $(3/8)$, while in bipolar format, it is $2 \times (3/8) - 1 = -(1/4)$. It is obvious that SN could not represent a digital number exactly. The longer the bitstream, the higher the accuracy. Even though there exists a slight loss in computation accuracy, the advantage of SC is the significant lower hardware cost for arithmetic calculation when compared with conventional methods [32]. In unipolar format, multiplication can be performed by a single AND gate, which greatly reduces the complexity of circuit design. SNs could be generated using SN generators (SNGs).

Due to the small hardware cost, low-power consumption, and high fault tolerance, SC has been applied to many applications to make a trade-off between hardware efficiency and computing performance, such as probability circuit design, network quantization, and ECC decoding [33], [34], [35], [36], [37]. In order to bring neural network computing into SC domain, a large number of SNGs are required to convert digital weights and inputs into bitstreams. It is a challenge for an SC-based neural network to achieve a lower computational latency and energy consumption compared with conventional designs [38].

III. SC-Based BNN

The implementation process and noteworthy details of StocBNN are introduced in this section. Gaussian random numbers are represented with the format of 0-1 bitstream, and a simplified but efficient GRNG is put forward. We prove the feasibility of the representation and demonstrate the dataflow in feed-forward propagation procedure. These construct the underlying theory for the successful application of StocBNN. It is surprising that when implementing BNN in SC domain, no additional SNGs are required, and the sampling operation could be also simplified.

A. Gaussian Random Number in Bitstream Format

In probability theory, the CLT establishes that in some situations, when independent random variables are added, their properly normalized sum tends toward a normal distribution even if the original variables themselves are not normally distributed. As a special case, the De Moivre–Laplace theorem states that the normal distribution may be used as an approximation to the binomial distribution under certain conditions [39]. Based on the De Moivre–Laplace theorem and CLT, we can draw the following corollary.

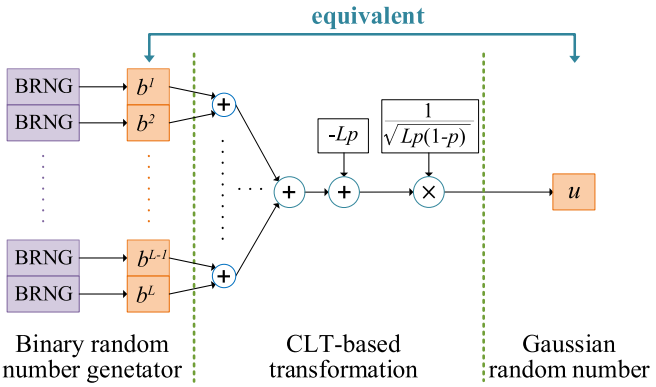

Fig. 3. Central limit theorem-based GRNG.

Corollary 1: Let $\{X_1, X_2, \ldots, X_L\}$ be a sequence of independent random variables that obey Bernoulli distribution and $E(X_k) = p(k = 1, 2, \ldots, L)$. If L is large enough, $((\sum_{k=1}^{L} X_k - Lp)/(Lp(1-p))^{1/2})$ fits standard Gaussian distribution $N(0, 1)$.

Fig. 3 demonstrates the computation procedure of CLT-based GRNG. The GRNG contains a series of binary random number generators (BRNGs) that could randomly generate 1 (or 0) with probability p (or $1 - p$). That is to say, $b^k \in \{0, 1\}(k = 1, 2, \ldots, L)$. After the CLT transformation, a Gaussian random number u is sampled corresponding to standard Gaussian distribution. In general, a 128-bit linear feedback shift register (LFSR) can be used for implementing BRNGs. By using a parallel counter, the number of 1's in the output of LFSR can be converted to a digital Gaussian random number based on CLT [40]. Usually, the parallel counter is implemented by adder tree. However, it requires 120 full adders for a 128-input parallel counter approximately, which consumes huge hardware cost. Motivated by this, the bitstream format Gaussian random numbers are proposed to address the challenge.

Based on CLT, a sequence of binary random numbers $\{b^1, b^2, \ldots, b^L\}$ and Gaussian random number u have the following relation:

$$u = f(b^1, b^2, \ldots, b^L) = \frac{\sum_{k=1}^{L} b^k - Lp}{\sqrt{Lp(1-p)}}. \tag{2}$$

It can be found from Fig. 3 and (2) that there exists an equivalent relation between the initial 0-1 bitstream $b^1 b^2, \ldots, b^L$ and the final Gaussian random number u. As aforementioned in Section II-D, the 0-1 bitstream in SC domain could be converted into a digital number. In unipolar format, 0-1 bitstream $b^1 b^2, \ldots, b^L$ in SC domain is equal to the number $((\sum_{k=1}^{L} b^k)/L)$ in binary radix-based computing domain. Next, we try to prove that $((\sum_{k=1}^{L} b^k)/L)$ is a Gaussian random number.

Based on probability theory, Gaussian distribution has a linear characteristic, which could be expressed as the following lemma.

Lemma 1: If $U \sim N(0,1)$, then $X = U * \sigma + \mu \sim N(\mu, \sigma^2)$.

Based on Lemma 1 and (2), we can get the following corollary.

Corollary 2: $((\sum_{k=1}^{L} b^k)/L)$ is also a Gaussian random number, which is sampled from Gaussian distribution $N(p, (((p(1-p))/L))$.

Proof: Equation (2) could be equivalently transformed into the following equation:

$$u \cdot \sqrt{\frac{p(1-p)}{L}} + p = \frac{\sum_{k=1}^{L} b^k}{L}. \tag{3}$$

Since u is a Gaussian random number sampled from $N(0,1)$, based on Lemma 1, it could be proved that $((\sum_{k=1}^{L} b^k)/L)$ is also a Gaussian random number.

On the basis of the SC method, $((\sum_{k=1}^{L} b^k)/L)$ represents the value of 0-1 bitstream $b^1 b^2, \ldots, b^L$. Thus, $b^1 b^2, \ldots, b^L$ is also a Gaussian random number not in digital format but in 0-1 bitstream format.

If 0-1 bitstream could participate in the subsequent feed-forward propagation, the complex transformation computation in CLT could be omitted. By this way, the parallel counter is not required, and large amount of hardware resources could be saved. As a result, the computing complexity, energy consumption, and hardware utilization can be significantly optimized.

B. SC-Based Feed-Forward Propagation

Representing Gaussian random numbers in 0-1 bitstream format could simplify the realization of GRNG. In order to enable bitstream format Gaussian random numbers flow through the BNN feed-forward propagation, the SC method should be employed in this procedure.

As introduced before, once a concrete neural network has been instantiated, BNN can perform inference using the same methods as a standard neural network. Based on neural network theory, the output of a neural network is calculated by (4). As for BNN, (4) would be performed several times

$$\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b}. \tag{4}$$

Here, we suppose that the input neural count is N, and the output neural count is M. Thus, the dimensions of $\boldsymbol{x}$, $\boldsymbol{y}$, W, and $\boldsymbol{b}$ are N, M, $M \times N$, and M.

The computation cost of vector addition could be neglected when compared with that of matrix-vector multiplication. So, we only consider $W\boldsymbol{x}$ and ignore the bias terms (i.e., addition between $W\boldsymbol{x}$ and $\boldsymbol{b}$).

Each element of $\boldsymbol{y}$, $\boldsymbol{y_i}(i = 1, 2, \ldots, M)$, is calculated as follows:

$$y_i = \sum_{j=1}^{N} w_{ij} x_j = \sum_{j=1}^{N} (u_{ij}\sigma_{ij} + \mu_{ij}) x_j. \tag{5}$$

Here, u_{ij} indicates a random number sampled from standard Gaussian distribution.

Associated with CLT-based GRNG, (5) can be reformulated as follows:

$$\begin{aligned} y_i &= \sum_{j=1}^{N} (u_{ij}\sigma_{ij} + \mu_{ij}) x_j \\ &= \sum_{j=1}^{N} \left(\frac{\sum_{k=1}^{L} b_{ij}^k - Lp}{\sqrt{Lp(1-p)}} \sigma_{ij} + \mu_{ij} \right) x_j. \end{aligned} \tag{6}$$

In Section III-A, it is pointed out that in unipolar format, 0-1 bitstream $b^1 b^2, \ldots, b^L$ is equivalent to $((\sum_{k=1}^{L} b^k)/L)$. Considering that $((\sum_{k=1}^{L} b^k)/L)$ does not appear in (6) explicitly, if $b^1 b^2, \ldots, b^L$ wants to participate in the feed-forward propagation, transformation should be performed as follows:

$$\begin{aligned} y_i &= \sum_{j=1}^{N} \left(\frac{\sum_{k=1}^{L} b_{ij}^k - Lp}{\sqrt{Lp(1-p)}} \sigma_{ij} + \mu_{ij} \right) x_j \\ &= \sum_{j=1}^{N} \left[\left(\frac{\sum_{k=1}^{L} b_{ij}^k}{L} \sqrt{\frac{L}{p(1-p)}} - \sqrt{\frac{Lp}{1-p}} \right) \sigma_{ij} + \mu_{ij} \right] x_j \\ &= \sum_{j=1}^{N} \left[\frac{\sum_{k=1}^{L} b_{ij}^k}{L} \sqrt{\frac{L}{p(1-p)}} \sigma_{ij} + \left(\mu_{ij} - \sqrt{\frac{Lp}{1-p}} \sigma_{ij} \right) \right] \\ &\quad \times x_j. \end{aligned} \tag{7}$$

Here, let $h_{ij} = ((\sum_{k=1}^{L} b_{ij}^k)/L)$, $\sigma'_{ij} = (L/(p(1-p)))^{1/2}\sigma_{ij}$, and $\mu'_{ij} = \mu_{ij} - (Lp/(1-p))^{1/2}\sigma_{ij}$. Equation (7) could be simplified as follows:

$$y_i = \sum_{j=1}^{N} (h_{ij}\sigma'_{ij} + \mu'_{ij}) x_j. \tag{8}$$

In (8), with the updated mean μ'_{ij} and variance ${\sigma'_{ij}}^2$, $((\sum_{k=1}^{L} b_{ij}^k)/L)$ could participate in the computing as a whole.

To realize the feed-forward propagation in SC domain, some minor transformations are made on (8). We divide the left-hand side and the right-hand side by two, as shown in (9). With this equation, the addition could be realized by a MUX. Thus, both the multiplication and the addition in BNN could be realized using the SC method

$$\frac{y_i}{2} = \frac{\sum_{j=1}^{N} h_{ij}\sigma'_{ij} x_j + \sum_{j=1}^{N} \mu'_{ij} x_j}{2}. \tag{9}$$

In the proposed StocBNN, h_{ij} is represented by 0-1 bitstream $b^1 b^2, \ldots, b^L$. μ'_{ij} and σ'_{ij} would also be converted into 0-1 bitstream format. With a well-trained BNN, the conversion of μ'_{ij} and σ'_{ij} only performs once. The dataflow of StocBNN is illustrated in Fig. 4 with a bitstream format input $\boldsymbol{x}$.

In Fig. 4, all the variables are stored in bitstream format except for $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}$, and $\bar{\boldsymbol{y}}$. The whole dataflow could be divided into four steps.

1) $\beta = \sigma' \ \& \ \boldsymbol{x}^T$. In this step, $\boldsymbol{x}^T$ performs the element-wise multiplication with each row of σ'. The dimension of the output, β, is the same as that of σ'.
2) $\gamma = \beta \ \& \ H$. This step performs element-wise multiplication between β and H.
3) $\eta = \mu' \ \& \ \boldsymbol{x}^T$. The operations in this step are similar to those in the first step.

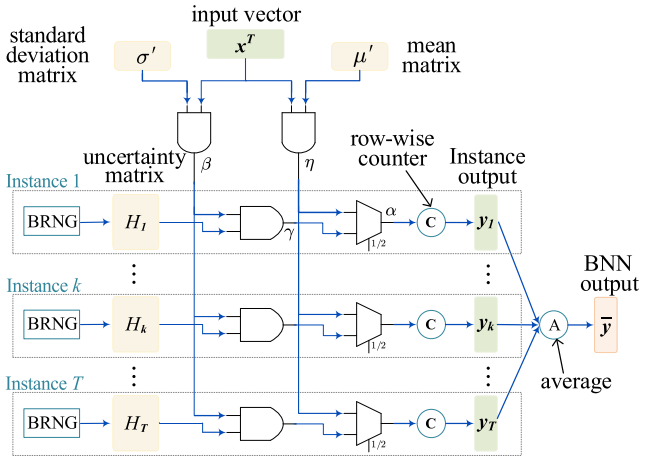

Fig. 4. Dataflow of StocBNN. Except for $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_T}$ and $\bar{\boldsymbol{y}}$, other variables are in bitstream format.

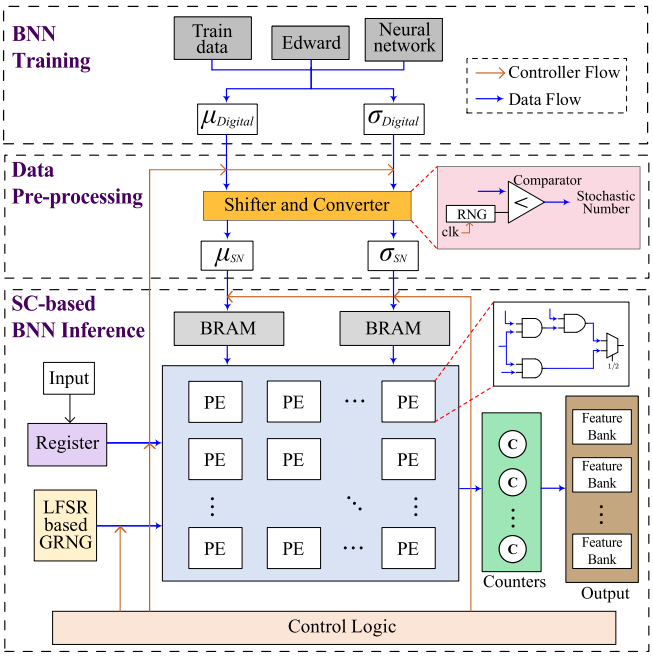

Fig. 5. Overview architecture of StocBNN.

4) $\alpha = ((\eta + \gamma)/2)$. The addition operation is always realized by MUX gates in the SC method. The select signal is a bitstream that contains half zeros and half ones. This is why we transform (8) to (9).

A line-wise count operation is used to accumulate the "1" in each row of α as the output of each concrete neural network [i.e., $\boldsymbol{y_k}(k = 1, 2, \ldots, T)$]. Subsequently, the BNN output $\bar{\boldsymbol{y}}$ is achieved with an average operation.

IV. Architecture of StocBNN

A. Overview

In this section, the overview architecture of the proposed StocBNN is introduced. As shown in Fig. 5, the whole architecture contains three steps: BNN training, data preprocessing, and SC-based BNN inference. In training phase, network parameters are obtained using the neural network structure and Edward framework mentioned earlier. Then, a shifter and converter are used to translate these well-trained parameters from digital domain to SC domain. In inference phase, LFSR-based GRNG, block memories (BRAMs), register, amount of process elements (PEs), counters, and controller are included. LFSR is used to randomly generate a series of zero or one, which are treated as the Gaussian random numbers. BRAMs are used to store the well-trained BNN parameters, including mean μ and standard deviation σ. The input will be stored and transferred in a shift register. PEs are utilized to perform feed-forward propagation. The feature output result is obtained by counters. Controller is used to make sure the correctness of timing.

B. LFSR-Based GRNG

In SC domain, the complex transformation in CLT is omitted. GRNG only needs to generate independent random number 0 or 1 based on Bernoulli distribution. In order to obtain adequate and high-quality SNs efficiently, linear feedback shift register (LFSR) is applied. An LFSR is usually implemented by a set of interconnected single-bit storage registers and a feedback network [41]. The outputs of n single-bit registers at time t are given by $\{q_{n-1}^{(t)}, \ldots, q_1^{(t)}, q_0^{(t)}\}$, and the next state of the registers is determined by a set of binary equations as (10) and (11)

$$\left\{\begin{array}{c} q_{n-1}^{(t+1)} = a_{n-1} \odot q_{n-1}^{(t)} \oplus \cdots \oplus a_1 \odot q_1^{(t)} \oplus a_0 \odot q_0^{(t)} \\ q_{n-2}^{(t+1)} = q_{n-1}^{(t)} \\ \vdots \\ q_0^{(t+1)} = q_1^{(t)} \end{array}\right\} \tag{10}$$

$$\boldsymbol{q}^{(t+1)} = A \odot \boldsymbol{q}^{(t)}. \tag{11}$$

Here, $a_i (i = 1, 2, \ldots, n)$ is binary coefficient, multiplication ($\odot$) is binary AND function, and addition ($\oplus$) is binary XOR function; $\boldsymbol{q}$ is the register state vector, and A is the transition matrix for one cycle.

We employ the multiple-bit skip-ahead method [42] to implement LFSR-based GRNG. When $\boldsymbol{q}$ is advanced k cycles, (11) can be expressed as follows:

$$\boldsymbol{q}^{(t+k)} = A \odot \boldsymbol{q}^{(t+k-1)} = A^k \odot \boldsymbol{q}^{(t)}. \tag{12}$$

Taking A^k as the transition matrix for one cycle, then LFSR can be advanced k steps in only one cycle. The outputs turn to be the following:

$$\boldsymbol{q}^{(t+1)} = A^k \odot \boldsymbol{q}^{(t)}. \tag{13}$$

In traditional LFSR, only the highest bit would be altered, and the left bits just shift by one position. In (13), the highest k bits are altered to reduce the correlation between generated data. For example, considering a 4-bit LFSR with coefficients $a_i = \{a3, a2, a1, a0\} = \{0, 1, 1, 0\}$, (13) is converted to (14) at $k = 3$

$$\left\{\begin{array}{c} q_3^{(t+1)} = q_3^{(t)} \oplus q_2^{(t)} \oplus q_1^{(t)} \\ q_2^{(t+1)} = q_3^{(t)} \oplus q_2^{(t)} \\ q_1^{(t+1)} = q_2^{(t)} \oplus q_1^{(t)} \\ q_0^{(t+1)} = q_3^{(t)} \end{array}\right\}. \tag{14}$$

It is clearly that q_3, q_2, and q_1 are altered. Compared with other methods, LFSR-based GRNG with skip ahead could achieve almost the same randomness with higher computational efficiency.

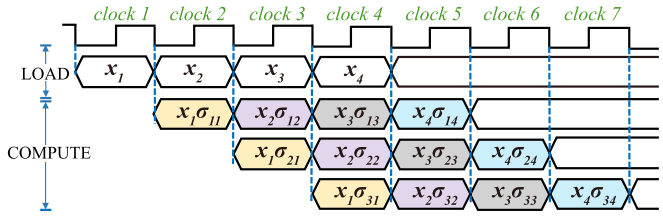

Fig. 6. Asynchronous parallel pipeline calculation technique: an example.

C. Asynchronous Parallel Pipeline Calculation Technique

As illustrated in Fig. 4, the input $\boldsymbol{x}$ performs multiplication with every row of σ and μ, and a row-wise count operation will be done next. It is easy to find that row is the basic unit in BNN inference phase. If T concrete neural networks are required, the input $\boldsymbol{x}$ will perform row-wise multiplication $M * T$ times (here, M is the count of output neural). Usually, $M * T > 1000$. If $\boldsymbol{x}$ performs all multiplication in parallel, there will be M fan-outs, which may increase the delay and power of logic circuit. Furthermore, the routing of this circuit is also a great challenge.

On the other hand, running time is an important indicator for neural networks when measuring network performance under the same conditions. Assuming that data loading and computing take one clock cycle each, if BNN is calculated row by row, it theoretically needs $2N$ clock cycles. A lot of time will be spent in waiting, which also causes a waste of hardware resources. In this work, a systolic array-aware asynchronous parallel pipeline technique is proposed to resolve these issues.

Systolic array is a homogeneous network of tightly coupled data processing units called nodes. Each node independently computes a partial result as a function of the data received from its upstream neighbors, stores the result within itself, and passes it downstream. Googles TPU is also designed around a systolic array [43].

Inspired by the characteristic of systolic array, an asynchronous parallel pipeline calculation technique is applied in our implementation. In order to make it easy to be understood, the proposed technique is described with the help of an example. Here, we suppose that the dimensions of $\boldsymbol{x}$ and σ are 4×1 and 3×4 ($M = 3$ and $N = 4$). The computing procedure based on asynchronous parallel pipeline calculation technique is shown in Fig. 6. In *clock 1*, $\boldsymbol{x_1}$ would be loaded. In *clock 2*, $\boldsymbol{x_2}$ is loaded, and $\boldsymbol{x_1}$ performs multiplication with σ_{11}. The fan-out of $\boldsymbol{x_1}$ is 2. In *clock 3*, $\boldsymbol{x_3}$ is loaded, and $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ perform multiplication, respectively, with σ_{21} and σ_{12}. The fan-out of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ is 2. Also, the multiplication between $\boldsymbol{x}$ and σ is finished after seven $(N + M)$ clock cycles, which is shorter than eight $(2N)$ clock cycles. When N is much larger than M, the advantage is obvious. In each clock, the fan-out of $\boldsymbol{x_i}(i = 1, \ldots, 4)$ is 2, which is less than M.

It is worth to point out two advantages of this technique. First, it could reduce the routing difficulties and the hardware resource utilized by routing, because the fan-out of each input element is reduced from M to 2. Second, it could improve the process speed of system because of the pipeline technique. Based on the experimental results, the system with this technique could work at 275 MHz. Otherwise, the system could only work around 100 MHz.

TABLE I
ACCURACY COMPARISON BETWEEN DIGITAL DOMAIN BNN (DIGTBNN) AND SC DOMAIN BNN (STOCBNN)

	Dataset	DigtBNN	StocBNN		
			32	64	128
4-FC	MNIST	98.29%	97.18%	97.2%	97.51%
	Fashion-MNIST	90.02%	87.64%	87.84%	88.13%
LeNet-5	MNIST	99.25%	98.39%	99%	99.15%
	Fashion-MNIST	99.36%	99.14%	99.28%	99.34%

V. EXPERIMENTS

Several experiments are performed to verify the efficiency of StocBNN in different dimensions. The experimental results are presented in this section.

A. Experiments Setup

This work focuses on improving BNN inference performance using the SC method. Several experiments have been done as the following three aspects: 1) learning accuracy comparison between conventional binary radix-based computing domain BNN (i.e., digital BNN) (DigtBNN) and StocBNN with software simulation; 2) the evaluation of hardware performance and generated Gaussian random number quality of LFSR-based SC domain GRNG; and 3) FPGA implementation and comparison between DigtBNN and StocBNN in terms of hardware performance.

B. Learning Accuracy

We use famous datasets of MNIST [44] and Fashion-MNIST [45] to evaluate learning accuracy in classification task. The concrete neural network count T is set as 100. Experiments are simulated with Python, and DigtBNN is implemented for comparison. Considering the issue of compounding errors over multiple layers [46], we implement the first layer with the SC method in inference process, and the remaining layers are implemented in digital domain. DigtBNN is performed with number format of 32-bit floating point.

StocBNN is first evaluated with a 784-200-200-10 configuration structure, and all layers are fully connected (named 4-FC). The BNN is pretrained using Edward, and ReLU is taken as the activation function. Experimental results of the 4-FC structure are demonstrated in the second and third rows of Table I, with a total of 10 000 test data. In addition, test accuracy of DigtBNN is shown in the third column. Columns 4–6 state the accuracy of StocBNN with a bitstream length of 32, 64, and 128, respectively. It could be found that as the increase of bitstream length, the testing accuracy increases. When the bitstream length is 128, there is only a slight decrease in an accuracy of about 0.78% compared with DigtBNN in MNIST and 1.89% in Fashion-MNIST.

The famous network structure of LeNet-5 [47] is also adopted to evaluate the efficiency of StocBNN when involving CNN architectures. The experimental results are demonstrated in the last two rows of Table I. The SC method is also implemented in the first convolution layer to avoid compounding errors. It can be seen that StocBNN achieves

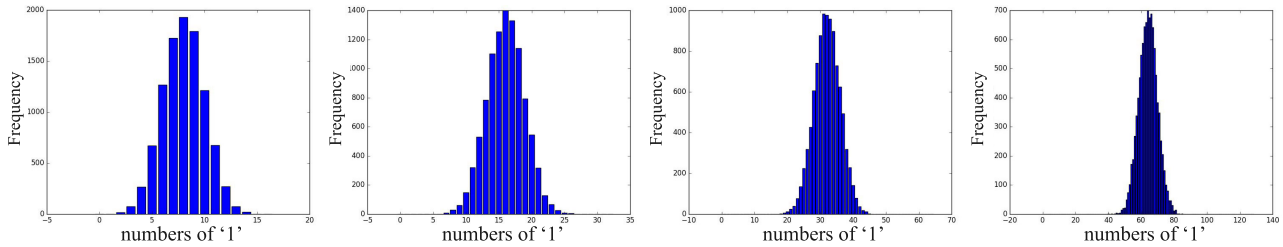

Fig. 7. Simulation results of LFSR-based SC domain GRNG. (a) Bitstream length = 16. (b) Bitstream length = 32. (c) Bitstream length = 64. (d) Bitstream length = 128.

TABLE II
NORMALITY TEST OF LFSR-BASED SC DOMAIN GRNG

	Build-in GRNG	LFSR-based GRNG			
		16	32	64	128
Pass Rate (%)	95.28	76.35	84.3	89.87	93.02

TABLE III
HARDWARE PERFORMANCE OF GRNG

Parameter	CLT-based GRNG	LFSR-based GRNG
Power(W)	0.011	0.002
# of LUT	132	28
# of Register	146	135
# of Slice	44	26

great performance in the LeNet-5 structure. When the bitstream length is 128, there is only a slight decrease in accuracy with about 0.1% and 0.02% in MNIST and Fashion-MNIST, respectively.

C. Evaluation of LFSR-Based GRNG

In Section IV-B, we have introduced LFSR-based GRNG with skip ahead. In this section, an LFSR-based SC domain GRNG is simulated with this technique in Vivado using Xilinx ZYNQ-7000 FPGA. Each GRNG could generate 128 bits (0 or 1) each cycle. As mentioned above, Gaussian random numbers play a very important role in BNN inference. In our experiments, 10 000 Gaussian random numbers are sampled with a bitstream length of 16, 32, 64, and 128 to evaluate the quality of GRNG. The histograms of these random numbers are drawn as in Fig. 7. The horizontal axis is the count of "1," and the vertical axis is the frequency. It could be seen that four group random numbers with different bitstream length could fit Gaussian distribution well.

Furthermore, normality test is also made for the generated random numbers using Kolmogorov–Smirnov test (K–S test), Shapiro–Wilk test (S–W test), and Lilliefors test to guarantee that those numbers fit Gaussian distribution [48]. We randomly construct 10 000 random number groups, and each group consists of 10–80 random numbers. The normality test is performed using Python, and the pass rate results are demonstrated in Table II. The second column shows the pass rate of random numbers that are sampled using the built-in GRNG of Python. It could be treated as the baseline. The following four columns are the results that random numbers are sampled using LFSR-based GRNG with the bitstream length of 16, 32, 64, and 128, respectively. The experimental results show that as the bitstream length increases, so does the pass rate. When the bitstream length reaches 128, the pass rate is only reduced by 2.26%.

We also make a comparison of hardware performance between LFSR-based GRNG and CLT-based GRNG. As shown in Table III, LFSR-based GRNG could almost reduce the power by 81.8% and save the hardware resources by 78.8%, 7.5%, and 40.9% in terms of LUT, register, and slice, respectively. The improvement of the energy and hardware efficiency is benefited from the omitting of complex transformation computation in CLT.

D. FPGA Implementation and Its Performance

In this section, StocBNN is implemented using Xilinx ZYNQ-7000 FPGA with Verilog language based on the architecture shown in Fig. 5. To facilitate StocBNN inference, inputs (i.e., images) are loaded from OFF-chip memory to ON-chip registers. In preprocessing phase, well-trained BNN parameters are converted from digital format to bitstream format and are stored in BRAM for the purpose of inference. LFSR-based GRNG with skip ahead is realized based on (13). In our experiment, the bit width of LFSR is set to be 128, and k in (13) is set to be 16. This allows for the generation of 128 0-1 numbers in a single inference per neuron to ensure sufficient randomness. Furthermore, traditional MAC units are replaced by PEs in SC domain, as illustrated in Fig. 5. Each PE consists of three AND gates and one MUX, and all the inputs of PE are in bitstream format. The inputs are then converted into digital domain using counters. Except for asynchronous parallel pipeline calculation technique, no other optimization techniques are applied in this work. Therefore, the hardware performance improvements demonstrated in this section are attributed to the proposed SC domain computing. Some effective FPGA acceleration strategies mentioned in Section I can also be used on StocBNN, such as the two-tier pipeline structure in [19] and neuron skipping strategy at hardware level in [26], for further improvement of the computing performance.

In order to demonstrate the efficiency of StocBNN, other three related cutting-edge works [19], [22], [24] with the same network structure are selected for comparison, collectively called DigtBNN. Table IV shows the hardware performance

TABLE IV
HARDWARE PERFORMANCE COMPARISON BETWEEN DIGTBNN AND STOCBNN

	DigtBNN			StocBNN		
	ASPLOS'18 [19]	TNNLS'20 [24]	DATE'20 [22]	This work		
Parameter Format	8-bit FixP	8-bit FixP	7-bit FixP	32	64	128
FPGA	ZYNQ-7000		ZYNQ-7020	ZYNQ-7000		
Max Freq. (MHz)	125	125	200	275		
Power (W)	0.926	0.823	2.76*	0.056	0.096	0.209
Throughput (μs/image)	64	64	7.6	114		
Energy Eff. (μJ/image)	59.264	52.7	21.09	6.384	10.944	23.826
# of LUT	10604	10148	37102	1363	2678	5555
# of Register	9595	8495	43268	1820	3094	6036
# of Slice	3411	3095	-	473	828	1633
# of DSP	100	52	200	0	0	0
# of BRAM	1	1	6	2	4	8

* The power is obtained based on the data shown in Fig.7 of [22] which is consumption of the FPGA board.

comparison of DigtBNN and StocBNN with the MNIST dataset in 4-FC. As shown in Table IV, StocBNN shows great performance in terms of hardware indicators. Both the methods in [19] and [24] are reimplemented by Verilog in order to make a fair comparison. Owing to the nonavailability of the project or well-trained model as open source, an assessment of the performance of [22] is reliant on the information provided in the original article. The performance of [19], [24], and StocBNN is reported based on Vivado synthesis results. Notably, the parameters of the BNN are represented differently across these studies. Parameters are employed an 8-bit fixed-point (FixP) representation in [19] and [24] and a 7-bit fixed-point representation in [22] and are transformed into 32-, 64-, and 128-bit SN in StocBNN (the second row in Table IV).

Benefited from the asynchronous parallel pipeline technique and simplified multiplication operation, the max frequency of StocBNN could achieve 275 MHz, while that of DigtBNN is only 200 Hz (the fourth row in Table IV). Power consumption reported by Vivado is shown in the fifth row. StocBNN achieves a 14.7× power consumption improvement in 32 bit, 8.6× in 64 bit, and 3.9× in 128 bit over DigtBNN [24].

In contrast to [19] and [24] that likewise conducted 100 times inference, StocBNN achieves a 1.78× increase in throughput and a 54.8% reduction in energy consumption with 128-bit bitlength. Compared with [22] who performs one time inference, StocBNN shows 15× increase in throughput, and the energy consumption of 128-bit StocBNN is increased by 12.9%.

Hardware resource utilization in terms of LUT, register, slice, DSP, and BRAM is shown in the last five rows. Since the parameters of BNN in SC domain are stored in bitstream format, more BRAMs are allocated in StocBNN. Despite BRAMs, the synthesized results show that StocBNN consumes much less hardware resources. Compared with DigtBNN, the LUT, register, slice, and DSP in StocBNN with 128-bit bitstream are reduced by 45.26%, 28.95%, 47.24%, and 100%, respectively.

It could be found from Table IV that operations in StocBNN are performed by LUTs. Due to that LUTs are generally six-input architecture but SC operations are two-input, FPGA is not the best implementation method for StocBNN. ASIC is another possible implementation approach. ASIC implementation can have more optimization strategies for SC operation than FPGA, while with the expense of flexibility and cost. For example, multiplications can be synthesized as AND gates that are more efficient than LUTs. For AI academic research, such as StocBNN, the model can be updated over time, and at this point, the FPGA implementation is more promising.

VI. CONCLUSION

In this article, BNN inference phase is performed in SC domain, which could reduce the power consumption with the cost of negligible learning accuracy decrease. Inspired by the CLT-based GRNG algorithm, the 0-1 sequence sampled from Bernoulli distribution can directly participate in the inference procedure using SC theory. With this strategy, the complex transformation in CLT-based GRNG is omitted. Furthermore, the asynchronous parallel pipeline calculation technique is proposed in computing block to enhance operation speed. Finally, both software simulation and hardware implementation are realized to evaluate the system performance. Software experiments show great potential of StocBNN in dealing with vision task. FPGA experimental results demonstrate that StocBNN consumes much less energy consumption and hardware resources. But, the high latency of StocBNN restricts its application in real-time tasks. In the future, we will explore the SC-based BNN with improved speed in more complicated network structures and nonvision tasks.

REFERENCES

[1] S. Pouyanfar et al., "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–36, Sep. 2019.
[2] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.
[3] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 604–624, Apr. 2020.

[4] F. Xing, Y. Xie, H. Su, F. Liu, and L. Yang, "Deep learning in microscopy image analysis: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4550–4568, Oct. 2018.

[5] Y. Li et al., "Deep learning for LiDAR point clouds in autonomous driving: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3412–3432, Aug. 2021.

[6] S. Dodge and L. Karam, "A study and comparison of human and deep learning recognition performance under visual distortions," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2017, pp. 1–7.

[7] F. Cui, Q. Cui, and Y. Song, "A survey on learning-based approaches for modeling and classification of human–machine dialog systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 4, pp. 1418–1432, Apr. 2021.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[9] H. W. Lin, M. Tegmark, and D. Rolnick, "Why does deep and cheap learning work so well?" *J. Stat. Phys.*, vol. 168, pp. 1223–1247, Sep. 2017.

[10] H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li, "Improving deep neural networks using softplus units," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–4.

[11] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 427–436.

[12] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with Bernoulli approximate variational inference," 2015, *arXiv:1506.02158*.

[13] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei, "Deep probabilistic programming," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–18.

[14] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with TensorFlow: A review," *J. Educ. Behav. Statist.*, vol. 45, no. 2, pp. 227–248, 2020.

[15] J. Shi et al., "ZhuSuan: A library for Bayesian deep learning," 2017, *arXiv:1709.05870*.

[16] E. Bingham et al., "Pyro: Deep universal probabilistic programming," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 973–978, 2018.

[17] J. L. Ticknor, "A Bayesian regularized artificial neural network for stock market forecasting," *Exp. Syst. Appl.*, vol. 40, no. 14, pp. 5501–5506, 2013.

[18] J.-T. Chien and Y.-C. Ku, "Bayesian recurrent neural network for language modeling," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 361–374, Feb. 2016.

[19] R. Cai et al., "VIBNN: Hardware acceleration of Bayesian neural networks," *ACM Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, vol. 53, no. 2, pp. 476–488, 2018.

[20] K. Yang, A. Malhotra, S. Lu, and A. Sengupta, "All-spin Bayesian neural networks," *IEEE Trans. Electron Devices*, vol. 67, no. 3, pp. 1340–1347, Mar. 2020.

[21] Y. Hirayama, T. Asai, M. Motomura, and S. Takamaeda-Yamazaki, "A resource-efficient weight sampling method for Bayesian neural network accelerators," in *Proc. 7th Int. Symp. Comput. Netw. (CANDAR)*, Nov. 2019, pp. 137–142.

[22] H. Awano and M. Hashimoto, "BYNQNet: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA," in *Proc. IEEE/ACM Proc. Design, Autom. Test Eurpoe (DATE)*, Mar. 2020, pp. 1402–1407.

[23] H. Awano and M. Hashimoto, "B2N2: Resource efficient Bayesian neural network accelerator using Bernoulli sampler on FPGA," *Integration*, vol. 89, pp. 1–8, Mar. 2023.

[24] X. Jia, J. Yang, R. Liu, X. Wang, S. D. Cotofana, and W. Zhao, "Efficient computation reduction in Bayesian neural networks through feature decomposition and memorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 4, pp. 1703–1712, May 2020.

[25] Y. Lin et al., "Bayesian neural network realization by exploiting inherent stochastic characteristics of analog RRAM," in *IEDM Tech. Dig.*, Dec. 2019, pp. 6–14.

[26] Q. Wan and X. Fu, "Fast-BCNN: Massive neuron skipping in Bayesian convolutional neural networks," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 229–240.

[27] H. Fan, M. Ferianc, M. Rodrigues, H. Zhou, X. Niu, and W. Luk, "High-performance FPGA-based accelerator for Bayesian neural networks," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1063–1068.

[28] H. Fan et al., "FPGA-based acceleration for Bayesian convolutional neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 12, pp. 5343–5356, Mar. 2022.

[29] R. Dorrance, D. Dasalukunte, H. Wang, R. Liu, and B. Carlton, "Energy efficient BNN accelerator using CiM and a time-interleaved Hadamard digital GRNG in 22 nm CMOS," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2022, pp. 2–4.

[30] J. S. Malik and A. Hemani, "Gaussian random number generation: A survey on hardware architectures," *ACM Comput. Surveys*, vol. 49, no. 3, p. 53, 2016.

[31] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, p. 92, May 2013.

[32] A. Mondal and A. Srivastava, "Power optimizations in MTJ-based neural networks through stochastic computing," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2017, pp. 1–6.

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

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

[35] X. Jia, J. Yang, Z. Wang, Y. Chen, H. H. Li, and W. Zhao, "Spintronics based stochastic computing for efficient Bayesian inference system," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 580–585.

[36] R. Cai et al., "A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology," in *Proc. 46th Int. Symp. Comput. Archit.*, Jun. 2019, pp. 567–578.

[37] X. Jia, J. Yang, P. Dai, R. Liu, Y. Chen, and W. Zhao, "SPINBIS: spintronics-based Bayesian inference system with stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 789–802, Apr. 2020.

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

[39] H. M. Walker and M. Helen, "De Moivre on the law of normal probability," in *A Source Book Mathematics*, S. D Eugene Ed. New York, NY, USA: Dover, 1985.

[40] R. Andraka and R. Phelps, "An FPGA based processor yields a real time high fidelity radar environment simulator," in *Proc. MAPLT*, 1998, pp. 220–224.

[41] G. Marsaglia, "Random numbers fall mainly in the planes," *Proc. Nat. Acad. Sci. USA*, vol. 61, no. 1, pp. 25–28, Sep. 1968.

[42] L. Colavito and D. Silage, "Efficient FPGA LFSR implementation whitens pseudorandom numbers," in *Proc. ACM Symp. Field Program. Gate Arrays (FPGA)*, 2009, pp. 308–313.

[43] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. IEEE/ACM Int. Symp. Comput. Archit. (ISCA)*, Nov. 2017, pp. 1–12.

[44] C. C. Y. LeCun and C. J. Burges. (2010). *MNIST Handwritten Digit Database*. [Online]. Available: http://yann.lecun.com/exdb/mnist

[45] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

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

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

[48] N. M. Razali and Y. B. Wah, "Power comparisons of Shapiro–Wilk, Kolmogorov–Smirnov, Lilliefors and Anderson–Darling tests," *J. Stat. Model. Analyt.*, vol. 2, no. 1, pp. 21–33, 2011.

Xiaotao Jia (Member, IEEE) received the B.S. degree in mathematics from Beijing Jiao Tong University, Beijing, China, in 2011, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, in 2016.

From 2016 to 2019, he was a Post-Doctoral Researcher with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing, where he is currently an Associate Professor with the School of Integrated Circuit Science and Engineering. His current research interests include spintronic circuits, stochastic computing, Bayesian deep learning, and EDA.

Huiyi Gu received the B.S. degree in electronic information engineering from Beihang University, Beijing, China, in 2017, where she is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering.

Her research interests include Bayesian deep learning and computing-in-memory architecture.

Yuhao Liu received the B.S. degree in electronic information engineering from Beihang University, Beijing, China, in 2020, where he is currently pursuing the master' degree with the School of Electronic and Information Engineer.

His research interests include analog circuit design and neuron computing.

Jianlei Yang (Member, IEEE) received the B.S. degree in microelectronics from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2014.

He is currently an Associate with the School of Computer Science and Engineering, Beihang University, Beijing. From 2014 to 2016, he was a Post-Doctoral Researcher with the Department of ECE, University of Pittsburgh, Pittsburgh, PA, USA. His current research interests include computer architectures and neuromorphic computing systems.

Dr. Yang was the recipient of the First/Second place on ACM TAU Power Grid Simulation Contest in 2011/2012. He was a recipient of the IEEE ICCD Best Paper Award in 2013, the ACM GLSVLSI Best Paper Nomination in 2015, the IEEE ICESS Best Paper Award in 2017, and the ACM SIGKDD Best Student Paper Award in 2020.

Xueyan Wang (Member, IEEE) received the B.S. degree in computer science and technology from Shandong University, Jinan, China, in 2013, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018.

From 2015 to 2016, she was a Visiting Scholar with the University of Maryland, College Park, MD, USA. She is currently an Assistant Professor with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing. Her current research interests include processing-in-memory architectures, AI chip, and hardware security.

Weitao Pan (Member, IEEE) received the B.S. degree from the School of Technical Physics, Xidian University, Xi'an, China, in 2004, and the Ph.D. degree from the School of Microelectronics, Xidian University, in 2010.

He is currently an Associate Professor with the State Key Laboratory of Integrated Service Networks, Xidian University. His current research interests include VLSI design methods and post-silicon verification.

Youguang Zhang (Member, IEEE) received the M.S. degree in mathematics from Peking University, Beijing, China, in 1987, and the Ph.D. degree in communication and electronic systems from Beihang University, Beijing, in 1990.

He is currently a Professor with the School of Electronic and Information Engineering, Beihang University. His research interests include microelectronics and wireless communication. In particular, he recently focuses on the circuit and system codesign for the emerging memory and computing systems.

Sorin Cotofana (Fellow, IEEE) received the M.Sc. degree in computer science from Politehnica University of Bucharest, Bucharest, Romania, in 1984, and the Ph.D. degree in electrical engineering from the Delft University of Technology, Delft, The Netherlands.

He is currently with the Faculty of Electrical Engineering, Mathematics and Computer Science, Computer Engineering Laboratory, Delft University of Technology, Delft. He has coauthored more than 250 papers in peer-reviewed international journal and conferences, and received 12 best paper awards in international conferences. His current research interests include the following: 1) the design and implementation of dependable/reliable systems out of unpredictable/unreliable components; 2) aging assessment/prediction and lifetime reliability-aware resource management; and 3) unconventional computation paradigms and computation with emerging nanodevices.

Dr. Cotofana is currently the Editor in Chief of the IEEE TRANSACTIONS ON NANOTECHNOLOGY, an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS and the IEEE Circuits and Systems Society (CASS), the Distinguished Lecturer, and a Board of Governors Member.

Weisheng Zhao (Fellow, IEEE) received the Ph.D. degree in physics from the University of Paris Sud, Paris, France, in 2007.

In 2009, he joined the French National Research Center (CNRS) as a Tenured Research Scientist. Since 2014, he has been a Distinguished Professor with Beihang University, Beijing, China. He is currently a Professor with the School of Microelectronics, Beihang University. He has published more than 200 scientific articles in leading journals and conferences, such as *Nature Electronics*, *Nature Communications*, *Advanced Materials*, IEEE TRANSACTIONS, ISCA, and DAC. His current research interests include the hybrid integration of nanodevices with CMOS circuit and new nonvolatile memory (40-nm technology node and below), such as MRAM circuit and architecture design.

Dr. Zhao is currently the Editor-In-Chief for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPER.