INTEGRATION, the VLSI journal 89 (2023) 1–8

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

B^2N^2: Resource efficient Bayesian neural network accelerator using Bernoulli sampler on FPGA

Hiromitsu Awano [*], Masanori Hashimoto

Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku, 606–8501, Kyoto, Japan

ARTICLE INFO

Keywords:
Bayesian neural network
Uncertainty
Monte Carlo
FPGA accelerator

ABSTRACT

A resource efficient hardware accelerator for Bayesian neural network (BNN) named B^2N^2, Bernoulli random number based Bayesian neural network accelerator, is proposed. As neural networks expand their application into risk sensitive domains where mispredictions may cause serious social and economic losses, evaluating the NN's confidence on its prediction has emerged as a critical concern. Among many uncertainty evaluation methods, BNN provides a theoretically grounded way to evaluate the uncertainty of NN's output by treating network parameters as random variables. By exploiting the central limit theorem, we propose to replace costly Gaussian random number generators (RNG) with Bernoulli RNG which can be efficiently implemented on hardware since the possible outcome from Bernoulli distribution is binary. We demonstrate that B^2N^2 implemented on Xilinx ZCU104 FPGA board consumes only 465 DSPs and 81661 LUTs which corresponds to 50.9% and 14.3% reductions compared to Gaussian-BNN (Hirayama et al., 2020) implemented on the same FPGA board for fair comparison. We further compare B^2N^2 with VIBNN (Cai et al., 2018), which shows that B^2N^2 successfully reduced DSPs and LUTs usages by 50.9% and 57.9%, respectively. Owing to the reduced hardware resources, B^2N^2 improved energy efficiency by 7.50% and 57.5% compared to Gaussian-BNN (Hirayama et al., 2020) and VIBNN (Cai et al., 2018), respectively.

1. Introduction

Neural networks (NNs) have emerged to demonstrate surpassing human performances on several applications such as image recognition [1–3], object detection [4] or voice generation [5]. Encouraged by these successes, NNs expand their application to risk sensitive domains such as self-driving cars [6,7] and flight control [8]. However, for such safety critical applications, making accurate predictions are not enough, i.e., NNs are requested to provide confidence on their prediction. Hence, Bayesian neural networks (BNNs), which are variants of NNs that provide mathematically grounded ways for uncertainty estimation on their prediction [9,10], attract increasing attention to replace NNs in risk sensitive applications. Contrary to conventional NNs where parameters are fixed during inference, BNNs treat them as random variables each has own probability distributions. The outputs of BNNs are also given as probability variables, which enables us to estimate their uncertainty, i.e., the wide probability distribution indicates the large uncertainty on it. Due to massive amount of computation required for the BNN inference, its hardware accelerators are intensively studied [11–13].

BNN accelerators can be classified into two categories in terms of adopted algorithm: deterministic variational inference and Monte-Carlo (MC) inference. The deterministic variational inference algorithm has been recently developed in machine learning community [14,15]. The major advantage of this algorithm is to eliminate the need for MC and thus is computationally efficient. Since NN inference involves multiply-accumulate (MAC) operations, i.e., the outputs of preceding neurons are multiplied by the corresponding synaptic weights followed by accumulation, the accumulated activation follows a Gaussian distribution due to the central limit theorem (CLT). Knowing that the network activation approximately follows a Gaussian distribution, all we have to propagate are mean and variance of activations since a Gaussian distribution is fully described by using up to the second moments. FPGA-based sampling-free BNN accelerator has been proposed in [13], which demonstrated 4.07× higher throughput compared with the conventional BNN accelerator. To simplify implementation, their work replaced ReLU non-linearity with quadratic ones. Although their experiment on MNIST dataset demonstrated that replacing ReLU with

[*] Corresponding author.
E-mail address: awano@i.kyoto-u.ac.jp (H. Awano).

https://doi.org/10.1016/j.vlsi.2022.11.005
Received 6 July 2021; Received in revised form 20 March 2022; Accepted 6 November 2022
Available online 10 November 2022

quadratic non-linearity causes very limited accuracy drop, our reproduction experiment revealed that BYNQNet achieves the classification accuracy of only 11.7% on CIFAR10, which is unacceptably low.

The MC inference, i.e., repeating forward passes with sampling parameters from the corresponding distributions to approximate the complex distributions of activations, is more flexible in terms of network structures and are successfully applied for various applications [16–18]. Since NNs have huge number of synaptic weights, increasing throughput of Gaussian random number generator (GRNG) is of great interest when considering the acceleration of MC inference. VIBNN [11], which is a prior work in this direction, employed a CLT-based GRNG which generates Gaussian random numbers (GRNs) by adding together 128 random bits generated by a linear shift feedback register (LFSR). However, naive implementation of CLT-based GRNG requires an accumulator for each GRNG, which significantly increases hardware cost. Hence, VIBNN focused on the locality of LFSR, i.e., only the bits on tap locations are updated at each iteration, and proposed to exploit a tiny accumulator which keeps track the changes of random bits on tap locations. More recently, another approach to improve resource efficiency of GRNG has been proposed in [12], where the GRNs were generated by using an inverse transform sampling method. By exploiting the error tolerance of NNs to the reduced numerical precision, [12] adopted 6-bit GRNGs which can be efficiently implemented by using 6-input look-up-table (LUT) on Xilinx 7 series FPGA. As the result, [12] showed that a single GRNG can be implemented by using only 3 slice LUTs while VIBNN implemented on the same FPGA required 12 slice LUTs.

In this paper, to further improve the resource efficiency of the BNN accelerator, we propose a novel BNN accelerator on FPGA which exploits Bernoulli random number generator (BRNG) instead of GRNG. Our proposal is based on the observation that the MAC output follows Gaussian distribution according to CLT and thus we can choose arbitrary statistical distributions instead of Gaussian which are suitable for hardware implementation. Our proposed accelerator named B^{2}N^2, Bernoulli random number based Bayesian neural network accelerator, generates samples from Bernoulli distributions whose parameters are adjusted so that the means and variances of original Gaussian distributions are conserved. Contrary to VIBNN [11] or LUT-based GRNGs [12] which require dedicated accumulator or LUT on each GRNG, B^{2}N^2 exploits MAC operations performed during NN inference as a part of a GRNG. The difference between the conventional Gaussian random number based BNN (Gaussian-BNN) accelerators and B^{2}N^2 is illustrated in Fig. 1. Note here that a random number generator (RNG) of B^{2}N^2 composed by only a BRNG and an MUX while the Gaussian-BNN accelerator requires a GRNG, a multiplier, and an adder for shifting and scaling unit GRNs.

B^{2}N^2 implemented on ZCU104 FPGA board requires only 465 DSPs and 88040 LUTs, which correspond to 50.9% and 14.3% reduction compared to Gaussian-BNN accelerator [12] carefully reimplemented on the same FPGA board for fair comparison. We further demonstrate that B^{2}N^2 achieved energy efficiency of 72.9 images/J, which corresponds to 7.50% improvement.

Followings summarize the contributions of this paper.

- To the best of our knowledge, this is the first BNN accelerator on FPGA achieving a practical performance on CIFAR10 dataset which is more practical than MNIST used in the conventional works.
- B^{2}N^2 implemented on ZCU104 board achieves 50.9% and 14.3% reduction of DSP and LUT usage compared to Gaussian-BNN accelerator [12] reimplemented on the same board for fair comparison.

The rest of this paper is organized as follows. In Section 2, we provide the preliminaries required to introduce B^{2}N^2, followed by the detail of B^{2}N^2 in Section 3. The software implementation of B^{2}N^2 is compared with Gaussian-BNN in Section 5. Then, in Section 4, B^{2}N^2 on ZCU104 board is compared with the Gaussian-BNN. Finally, concluding remarks are provided in Section 6.

2. Preliminaries

2.1. Bayesian neural network

BNN is an extension of NNs that can model uncertainties of predictions. For a general Bayesian model, we are interested in finding the posterior distribution over the weights of L-layered network, $\boldsymbol{W} = (\boldsymbol{w}^1, \boldsymbol{w}^2, \dots, \boldsymbol{w}^L)$, given the training data, $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_M)$, and the target label, $\boldsymbol{Y} = (y_1, y_2, \dots, y_M)$. Here, $\boldsymbol{w}^l$ is the $N_l \times N_{l-1}$ weight matrix of lth layer having N_{l-1} input and N_l output nodes, $\boldsymbol{x}_i$ is a D-dimensional input vector, and M is the number of training samples. By using the Bayes' theorem, the posterior distribution can be represented by the combination of the likelihood and the prior distribution:

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

In practical applications, this posterior distribution is not tractable and hence the *variational inference* technique has been developed to approximate the posterior distribution, i.e., $P(\boldsymbol{W}|\boldsymbol{X}) \approx q(\boldsymbol{W}|\boldsymbol{\theta})$, where $q(\cdot|\cdot)$ is a variational posterior distribution and $\boldsymbol{\theta}$ is variational parameters.

For simplicity, a Gaussian distribution is usually employed for the variational posterior distribution. Hence, an (i, j)-element of $\boldsymbol{w}^l$ can be obtained by

$$w^l_{ij} = \mu^l_{ij} + \epsilon \cdot \log\left(1 + \exp\left(\rho^l_{ij}\right)\right), \tag{2}$$

where ϵ is a random variable sampled from the unit Gaussian distribution, and μ^l_{ij} and ρ^l_{ij} are the variational parameters. During inference, we perform the forward propagation repeatedly with randomly sampled network weights to approximate the statistical distribution of network outputs:

$$P(y|\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) \approx \frac{1}{N}\sum_{n=1}^{N} g(y, \boldsymbol{x}, \boldsymbol{W}^{(n)}), \tag{3}$$

where $g(\cdot,\cdot,\cdot)$ is the network function and $\boldsymbol{x}$ and $\boldsymbol{y}$ are network input and output, respectively. $\boldsymbol{W}^{(n)}$ is nth sample drawn from $q(\boldsymbol{W}|\boldsymbol{\theta})$. Using generated samples, we can evaluate the uncertainty of the classification by several metrics such as the entropy defined by

$$H[y|\boldsymbol{x},\boldsymbol{X},\boldsymbol{Y}] = -\sum_{c} P(y=c|\boldsymbol{x},\boldsymbol{X},\boldsymbol{Y}) \log P(y=c|\boldsymbol{x},\boldsymbol{X},\boldsymbol{Y}). \tag{4}$$

BNN is capable of reporting the uncertainty in the network output, which is the distinct advantage of BNN, but it involves a large amount of computation since the forward propagation is repeated N times to make each prediction. Hence, hardware accelerators for BNN inference have been intensively studied. Due to the frequent usage of GRNs in the BNN inference, developing high-performance yet efficient GRNGs have been of great interest for the hardware acceleration of BNN.

2.2. Gaussian random number generators

A typical GRNG is composed of two components: a uniform random number generator (URNG) and a converter that transforms the uniform random numbers (URNs) to GRNs. Contrary to cryptographic applications where randomnesses are the most important, quasi-random sequences are sufficient for BNN inference and thus an LFSR is commonly adopted for URNG.

For the conversion of URNs to GRNs, several algorithms have been proposed, which includes Box–Muller method [19], CLT-based method, and CDF inversion method. Box–Muller method has been one of the most popular method for GRNs generation. It applies a series of mathematical functions such as sine and cosine on URNs to obtain GRNs. The advantage of this method is that it has no branching or looping and hence it can produce uncorrelated GRNs every steps, which is suitable for implementation on GPU-like architecture where massive amount of arithmetic units are available.

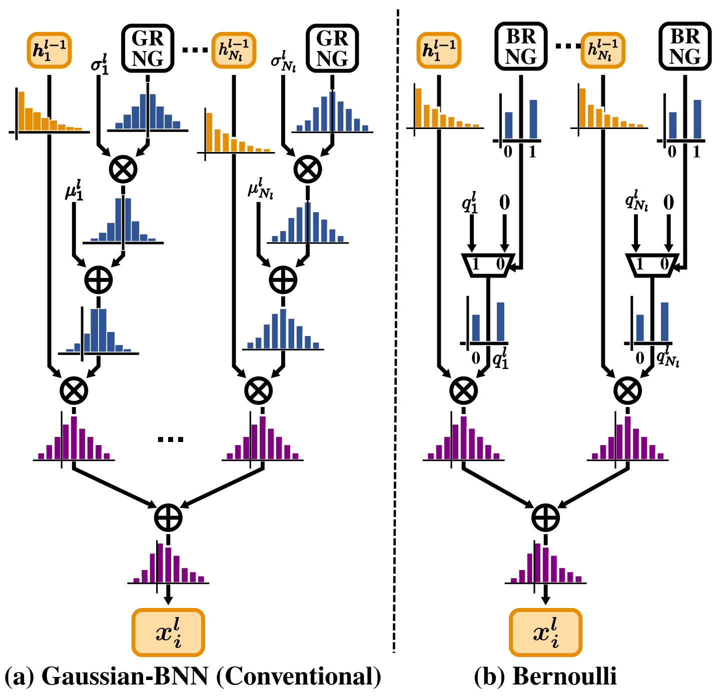

Fig. 1. (a) Gaussian-BNN accelerator and (b) proposed B^{2}N^2. Even if weights are sampled from Bernoulli distributions, the accumulating operations make the resulting activations to follow Gaussian distribution.

Considering the error tolerance of NNs, on the other hand, less accurate yet efficient methods are more desirable. Hence, VIBNN employed the CLT-based method [11] which exploits the fact that mean of n independent random variables will follow Gaussian distribution when n is sufficiently large. The major advantage of this method is that it requires only add and shift operations, which can be efficiently implemented with limited amount of hardware resources. However, the accumulation of hundreds of bits still requires considerable amount of hardware resources. Hence, [12] adopted the CDF inversion method where GRNs were generated by applying the inverse Gaussian CDF on URNs. Since computation of inverse Gaussian CDF on hardware is difficult due to its strong non-linearity, [12] introduced lookup table (LUT) approximation. Further, they demonstrated that the bit precision of GRNs can be safely reduced down to 6-bit so that the LUT can be efficiently implemented by using 6-input LUT primitives integrated on Xilinx 7 series FPGA.

Note here that above GRNGs generate uniform GRNs, i.e., the mean and variance are fixed to be zero and one, respectively. Hence, we still need additional adder/multiplier to generate GRNs having desired means and variances, which still requires huge hardware resources.

3. Proposed method

3.1. B^{2}N^2 architecture overview

In this section, we propose an efficient BNN accelerator named B^{2}N^2. Fig. 2 illustrates the entire architecture of B^{2}N^2 which consists of on-chip BRAMs and processing elements (PEs). We employed a streaming pipelining architecture, where each PE is responsible for the computation of single convolutional layer, to increase the inference throughput.

3.2. PE design

Each PE has an *im2col* unit, weight generators (WGs), and a matrix multiplication (MM) unit. Input feature maps are given in channel-major manner to avoid data duplication while realizing continuous accessing pattern for increased inter PE communication bandwidth [20]. The im2col unit extract image patches from the input feature stream and rearrange pixels so that the convolutional operations can be performed with the MM unit [21]. The im2col unit is implemented by using a shift register and a set of multiplexer units (MUXes). The shift register sequentially reads a pixel value from the input feature map stream while shifting out the last pixel. Then, MUXes select appropriate items from the shift register to form an internal stream of input pixels fed to the MM unit. The MM unit has multiple pairs of a multiplier and an accumulator each of which is responsible for computing a single output channel. The weight parameters are also rearranged in channel-major manner and stored in on-chip BRAMs.

Listing 1 shows the pseudo-code of PE for convolutional layer, where kernels are convolved over the input image with specific stride and padding. When performing high-level synthesis of code containing loops, pipeline directives can be added to significantly improve throughput. However, when adding pipeline directives to nested loops, the inner loops are automatically unrolled completely, and there is a risk that they will not fit into the limited FPGA resources. Therefore, in this study, we chose to fully expand only the innermost loop6. Moreover, since extra clock cycles are required to enter and exit loops, we flattened the loops. Loop flattening refers to the conversion of multiple loops into shallowly nested loops. For the description in the Listing 1, flattening Loop 1 to Loop 5 yields the description in the Listing 2, which is fed to Vivado_HLS to obtain the RTL of PEs.

Listing 1: Pesudo-code of PE design for convolutional layer

```
Loop1: for i in 1...input_channel // input image channel
 Loop2: for j in 1...width // picture width
  Loop3: for k in 1...height // picture height
   Loop4: for l in 1...filter_size_x // filter x direction
    Loop5: for m in 1...filter_size_y // filter y direction
     Loop6: for n in 1...output_channel // output image channel
      output[j][k][n] += weights[i][l][m][n] * input[j][k][i];
     end
    end
   end
  end
 end
end
```

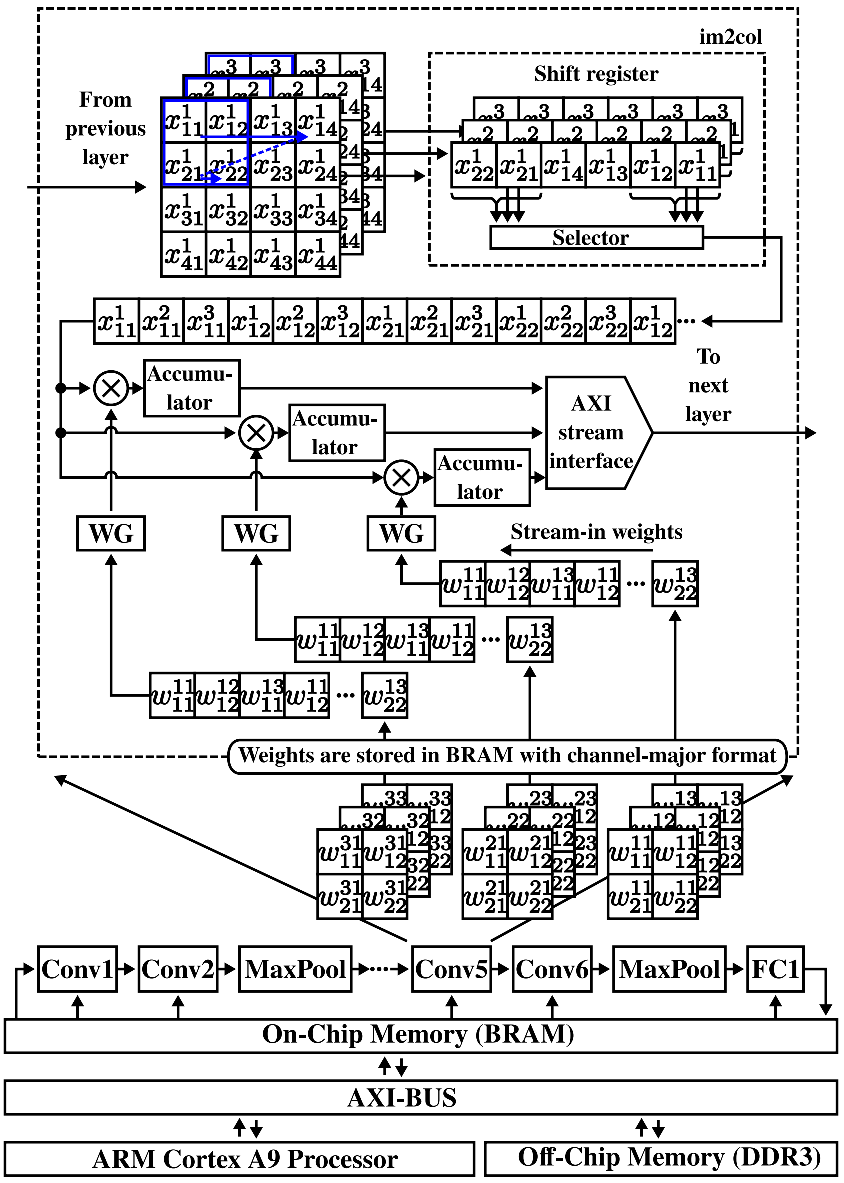

Fig. 2. Proposed B^2N^2 accelerator.

Listing 2: Modified Pseudo-code of PE design for convolutional layer

```
i=1; j=1; k=1; l=1; m=1;
Loop1: for ii in 1...input_channel * width * height *
                           filter_size_x * filter_size_y
#pragma HLS PIPELINE
 Loop2: for n in 1...output_channel//output image channel
  output[j][k][n] += weights[i][l][m][n] * input[j][k][i];
 end
 m++;
 if m = filter_size_y:
  l++;
  if l = filter_size_x:
   k++;
   if k = height:
    j++;
    if j = width:
     i++;
     j = 0;
    end
   end
  end
 end
end
```

3.3. Bernoulli random number generators

The novelty of B^2N^2 is to exploit BRNs for BNN inference while most conventional accelerators require GRNs. As discussed in Section 2.2, there are several ways to generate GRNs on hardware but all of those approaches suffer from large hardware footprint. For example, VIBNN samples GRNs by exploiting the CLT approximation, which requires accumulators for counting number of 1's in the LFSR. Authors of [12] introduced an LUT-based inverse transform sampling where LUTs precision are carefully optimized so that they fit Xilinx 7 series 6-input LUT primitives. However, it still requires a multiplier and an adder to shift and scale uniform GRNs so that they have intended means and variances, which leads to a significant hardware overhead.

To solve above mentioned problems, B^2N^2 employs BRNs which can be efficiently generated on hardware. One may argue that using non-Gaussian random numbers may violate the assumption of BNN that the weights are sampled from Gaussian distributions. However, according to CLT, even if the weights are sampled from non-Gaussian

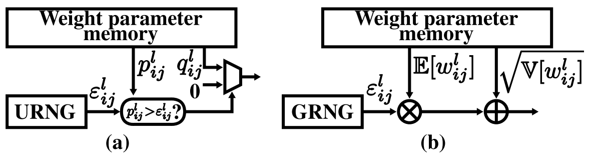

Fig. 3. (a) Weight generators of B^{2}N^2 and (b) that of Gaussian-BNN accelerator [12].

distributions, the MAC result follows a Gaussian distribution when the number of neurons is sufficiently large. Let us first review the algorithm for neural network inference. Neural network is composed of a series of linear and non-linear transformations. Given inputs for lth layer, $\boldsymbol{h}^{l-1} = \left(h_1^{l-1}, h_2^{l-1}, \ldots, h_{N_l}^{l-1}\right)$, the inputs are firstly applied a linear transformation:

$$x_i^l = b_i^l + \sum_{j=1}^{N_{l-1}} w_{ij}^l h_j^{l-1}, \tag{5}$$

where $\boldsymbol{b} = \left(b_1^l, b_2^l, \ldots, b_{N_l}^l\right)$ is the bias and $\boldsymbol{x}^l = \left(x_1^l, x_2^l, \ldots, x_{N_l}^l\right)$ are the lth layer pre-activations and $l-1$th layer post-activations, respectively. Then, a non-linear transformation $\phi(\cdot)$ is applied for $\boldsymbol{x}^l$ to yield the layer output $\boldsymbol{h}^{l+1}$, which is used as inputs for $l+1$th layer. Eq. (5) shows that x_i^l is given by a sum of independent and identically distributed (i.i.d) random variables and hence, owing to CLT, x_i^l follows a Gaussian distribution at the limit of infinite neurons, i.e., $N_{l-1} \to \infty$. The means and variances of $\boldsymbol{x}^l$ are given by:

$$\mathbb{E}[x_i^l] = b_i^l + \sum_{j=1}^{N_{l-1}} \mathbb{E}[w_{ij}^l]\mathbb{E}[h_j^{l-1}], \tag{6}$$

$$\mathbb{V}[x_i^l] = \sum_{j=1}^{N_{l-1}} \left\{ \left(\mathbb{V}[w_{ij}^l] + \mathbb{E}[w_{ij}^l]^2 \right) \mathbb{V}[h_j^{l-1}] + \mathbb{V}[w_{ij}^l]\mathbb{E}[h_j^{l-1}]^2 \right\}, \tag{7}$$

where $\mathbb{E}[\boldsymbol{x}]$ and $\mathbb{V}[\boldsymbol{x}]$ indicate the mean and variance of $\boldsymbol{x}$, respectively. Eqs. (6) and (7) show that as long as $\mathbb{E}[w_{ij}^l]$ and $\mathbb{V}[w_{ij}^l]$ are preserved, arbitrary distributions can be used for w_{ij}^l without changing the statistical properties of pre-activations.

BRNs are the simplest kind of RNs which takes “1” with probability p and “0” otherwise. Let X be a BRN, then $\mathbb{E}[X] = p$ and $\mathbb{V}[X] = p(1-p)$, respectively. Note that our goal is to replace GRNs with BRNs without interfering their means and variances. However, since Bernoulli distribution is a single parameter distribution, their means and variances cannot be controlled separately. Hence, we introduce an auxiliary parameter q and define a new random variable $Y = qX$. Then, the mean and variance of Y are given by $\mathbb{E}[Y] = pq$ and $\mathbb{V}[Y] = p(1-p)q^2$, respectively. To preserve $\mathbb{E}[w_{ij}^l]$ and $\mathbb{V}[w_{ij}^l]$, $\mathbb{E}[w_{ij}^l] = \mathbb{E}[Y]$ and $\mathbb{V}[w_{ij}^l] = \mathbb{V}[Y]$ should be held. Hence, we obtain

$$p_{ij}^l = \frac{\mathbb{E}[w_{ij}^l]^2}{\mathbb{E}[w_{ij}^l]^2 + \mathbb{V}[w_{ij}^l]}, \tag{8}$$

$$q_{ij}^l = \frac{\mathbb{E}[w_{ij}^l]^2 + \mathbb{V}[w_{ij}^l]}{\mathbb{E}[w_{ij}^l]}, \tag{9}$$

where p_{ij}^l and q_{ij}^l are parameters for w_{ij}^l. Eqs. (8) and (9) indicate that BRNs multiplied by an appropriately selected constant value can be adopted for BNN inference without interfering the statistical characteristics of output distribution. Note here that both Gaussian-BNN and B^{2}N^2 requires two precomputable variational parameters, i.e., $\mathbb{E}[w_{ij}^l]$ and $\mathbb{V}[w_{ij}^l]$ for Gaussian-BNN and p_{ij}^l and q_{ij}^l for B^{2}N^2, and hence both method requires the same memory footprint.

Fig. 3(a) shows the WG of B^{2}N^2, which is composed by a URNG, a comparator, and a multiplexer (MUX). URNG generates a uniform random value ε_{ij}^l, which is compared with p_{ij}^l provided by the weight parameter memory. If $p_{ij}^l > \varepsilon_{ij}^l$, MUX outputs q_{ij}^l and “0” otherwise. For the comparison, WG of [12] is shown in Fig. 3(b) where weight are generated by applying scaling and shifting on the uniform GRN generated by GRNG. Obviously, WG of B^{2}N^2 does not require multiplier, which contributes to the reduction of hardware cost.

4. Hardware implementation and measurement flow

4.1. Implementation flow

The overall flow of implementing B^{2}N^2 on ZCU104 is shown in Fig. 4. Firstly, B^{2}N^2 is implemented and trained by using TensorFlow probability framework. After the training, the means and variances of weights are extracted as a Numpy array format. Then, in-house parameter converter computes Eqs. (8) and (9) and generates a C++ header file.

Whole circuits of B^{2}N^2 are designed using C++ language. Since floating-point operations require huge cost when implemented on FPGA [22], every numerical operations are replaced by 8-bit fixed-point arithmetics. For the ease of implementation, we adopted “ap_fixed” template which handles fractional arithmetic. Xilinx Vivado toolchain is used to generate a “.bit” FPGA configuration file. The implemented B^{2}N^2 is controlled via Python scripting language using PYNQ framework.

Listing 3: Python code executed on PYNQ board

```
from pynq import Overlay
ol = Overlay('design.bit') # configure FPGA
registers = ol.HLS_accel_0.register_map
registers.CTRL.AP_START = 1 # start B2N2 accelerator
registers.CTRL.AUTO_RESTART = 1
rails = pynq.get_rails()
# power measurement
recorder = pynq.DataRecorder(rails['INT'].power)
with recorder.record(0.1):
    ol.axi_dma_0.sendchannel.transfer(ibuff) # transfer images
    ol.axi_dma_0.recvchannel.transfer(obuff) # load results
    ol.axi_dma_0.sendchannel.wait() # wait DMA to send data
    ol.axi_dma_0.recvchannel.wait() # wait DMA to receive data
```

4.2. Measurement flow

The code shown in Listing 3 is executed to measure image classification accuracy and power consumption. We measured power consumption of FPGA part by using PMBus power system embedded on ZCU104 board. During measurement, 10k test images are continuously applied to the B^{2}N^2 accelerator to observe the averaged power consumption. Since MC sampling involves random procedure, we repeated every experiment 10 times to investigate the impact of randomness.

5. Experimental results and discussion

5.1. Experimental setup

We compare B^{2}N^2 with Gaussian-BNN [12] and VIBNN [11] which are reimplemented on the same FPGA board for the fair comparison. Gaussian-BNN, VIBNN, and B^{2}N^2 are applied on the following image classification tasks to evaluate their performances on practical tasks.

MNIST MNIST dataset is a collection of 70k images of 28 × 28 grayscale hand written digit, where 60k images are used for model training while the rest of 10k images are for testing [23]. In our experiment, 28 × 28 image was resized to 32 × 32 to align the image size with the CIFAR10 dataset described next.

CIFAR10 CIFAR10 consists of 60k 32 × 32 pixel color images of animals and vehicles, of which 50k are used for training and the remaining 10k for testing [24].

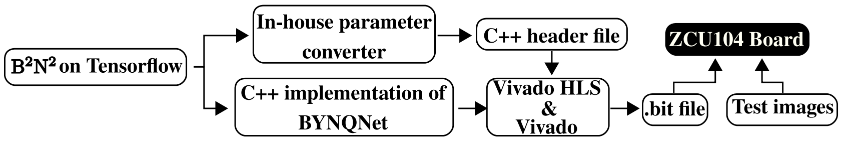

Fig. 4. Design flow of B^2N^2 accelerator.

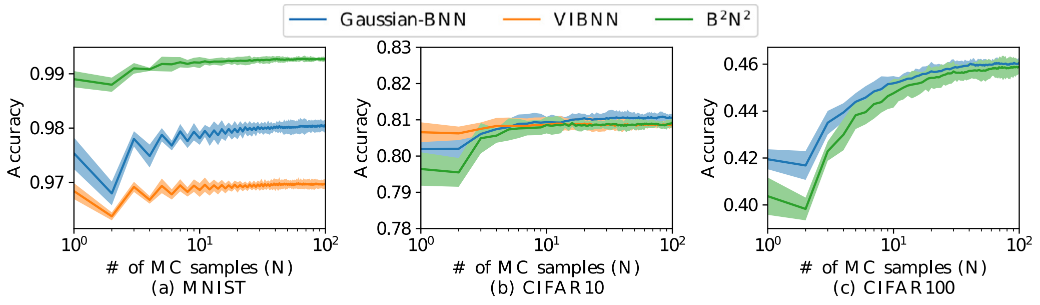

Fig. 5. Inference accuracies of Gaussian-BNN, VIBNN, and B^2N^2 as a function of MC samples.

Table 1
Network architecture.

Layer	Input Fmaps	Output Fmaps	Output Dim
Conv1	3	32	32
Conv2	32	32	32
MaxPool	32	32	16
Conv3	32	64	16
Conv4	64	64	16
MaxPool	64	64	8
Conv5	64	128	8
Conv6	128	128	8
MaxPool	128	128	4
Dense	4 × 4 × 128	10 (for MNIST and CIFAR10) 100 (for CIFAR100)	1

CIFAR100 This dataset is similar to CIFAR10 except that it has 100 classes containing 600 images. Among 600 images per class, 500 images are used for training and the remaining 100 images are used for testing. Since CIFAR100 requires 100 dimensional logits to be predicted, which increases the number synaptic of parameters, VIBNN could not be implemented in the ZCU104 due to its large circuit size. Therefore, we compared two methods for CIFAR100: Gaussian-BNN and B^2N^2.

For our experiment, we employed a VGG-like architecture which consists of six convolutional layers, three max pooling layers, and one densely-connected layer. 3 × 3 kernel filters are used for all convolutional layers. Table 1 shows the network architecture.

5.2. Inference accuracy

CLT assumes that (1) the number of neurons are large enough and that (2) each pixel value of input feature map is stochastically independent. However, these requirements may not hold for the practical neural network architecture and hence we firstly investigated the inference accuracy degradation caused by replacing GRNs with BRNs. Fig. 5 shows the inference accuracy of Gaussian-BNN, VIBNN, and B^2N^2 as a function of MC samples. Note here that since MC sampling involves random procedure, we repeated every experiment 10 times to investigate the impact of randomness. Note also that, for the experiments with CIFAR100 dataset, only the results for Gaussian-BNN and B^2N^2 are shown since VIBNN could not be implemented on ZCU104 due to its large circuit size. The bold lines and the shaded regions show the averages and 95% confidence intervals of 10 trials. We notice that the inference accuracies slightly improve by increasing MC samples. We also see that the accuracies of B^2N^2 are almost the same or slightly higher than Gaussian-BNN and VIBNN.

5.3. Uncertainty estimation

We further investigate the reliability of the estimated uncertainty by using the precision–recall (PR) metric [25]. First, we compute the uncertainty for each classification with Eq. (4) and rank images according to the associated uncertainty. Then, the testing accuracy is evaluated using only α-portion of images having lower uncertainty, while the others are discarded. The curve shows how classification accuracy changes as we discard images classified with lower confidence than different percentile thresholds.

Fig. 6 shows the PR curves of Gaussian-BNN, VIBNN, and B^2N^2 for different MC samples (N). Again, the shaded regions show the 95% confidence intervals. If the classifier is reliable, we should observe the monotonically decreasing PR curve. For Gaussian-BNN, VIBNN, and B^2N^2, the accuracies monotonically decrease, which indicates that the uncertainties estimated by these methods well correlate to misclassifications. We also notice that by increasing N, PR curves move to the upper. For example, Fig. 6(c) shows that if we take only the 20% of prediction results having lower uncertainties, the classification accuracy reaches over 90%.

To quantitatively compare the reliabilities of reported uncertainties, we compute the area under each PR curve (AUC) and summarize the result in Fig. 7. The shaded regions again show the 95% confidence intervals. A high AUC indicates that a classifier outputs accurate results while suppressing the misclassification and hence the AUC of the perfect classifier will be 1.0. Studying Fig. 7, we notice that AUC monotonically increases as a function of MC samples (N). We again see that AUCs of B^2N^2 are almost the same or slightly higher than Gaussian-BNN and VIBNN, which validate the use of BRNs instead of GRNs.

5.4. Resource utilization and energy consumption

Finally, we compare the resource utilization and energy consumption of Gaussian-BNN, VIBNN, and B^2N^2 when they are implemented on ZCU104 board. As shown in Table 2, B^2N^2 can be implemented by using only 465 DSPs and 89312 LUTs, which corresponds to 50.9% and 14.3% reduction compared to Gaussian-BNN [12]. Further, compared to VIBNN [11], B^2N^2 successfully reduced DSPs usage by 50.9% and LUTs usage by 57.9%. Owing to these hardware resource reductions, B^2N^2 achieved energy efficiency of 100.3 Images/J, which corresponds to 7.50% and 57.5% improvements from Gaussian-BNN and VIBNN, respectively.

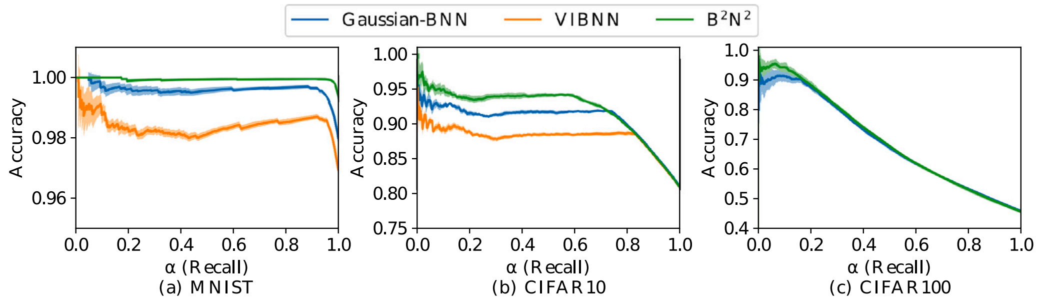

Fig. 6. Precision–recall (PR) curves of Gaussian-BNN, VIBNN, and B^{2}N^2 as a function of MC samples.

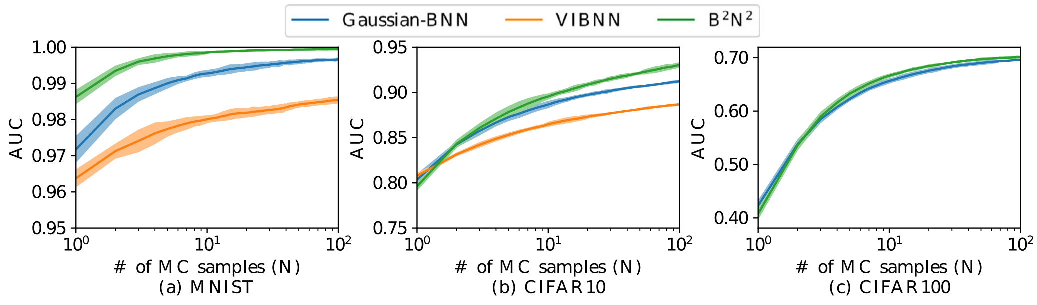

Fig. 7. Area under the precision–recall curves (AUC) of Gaussian-BNN, VIBNN, and B^{2}N^2 as a function of MC samples.

Table 2
Comparison with Gaussian-BNN accelerator.

	Gaussian-BNN [12]	VIBNN [11]	B^{2}N^2 (This work)
FPGA	Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC		
Clock (MHz)	300		
# of LUTs	101560/230400 (44.1%)	194106/230400 (84.3%)	81661/230400 (35.4%)
# of DSPs	929/1728 (53.8%)	929/1728 (53.8%)	465/1728 (26.9%)
Registers	117632/460800 (25.5%)	244436/460800 (53.1%)	89312/460800 (19.3%)
BRAM [kB]	1184/1404 (84.3%)	1109/1404 (79.0)	1370/1404 (97.6%)
Throughput (Images/s)	300.4	254.8	300.4
Energy efficiency on CIFAR10 (Images/J)	93.3	63.7	100.3

6. Conclusion

B^{2}N^2, an efficient BNN accelerator on FPGA, was proposed. Owing to CLT, we showed that arbitrary distributions can be used for sampling variational parameters as long as means and variances are hold. To reduce hardware resources, B^{2}N^2 employed BRNs which are the simplest kind of RNs taking either “1” or “0”. B^{2}N^2 implemented on ZCU104 FPGA board consumes only 465 DSPs and 81661 LUTs, which corresponds to 50.9% and 14.3% reductions compared to the Gaussian-BNN [12] implemented on the same FPGA board. Further, compared to VIBNN [11], B^{2}N^2 successfully reduced DSPs usage by 50.9% and LUTs usage by 57.9%. Owing to the reduced hardware resources, B^{2}N^2 improved energy efficiency by 7.50% and 57.5% compared to Gaussian-BNN and VIBNN, respectively.

CRediT authorship contribution statement

Hiromitsu Awano: Conceptualization, Methodology, Software, Visualization, Writing. **Masanori Hashimoto:** Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partially supported by Japan Science and Technology Agency (JST), PRESTO Grant No. JP-MJPR18M1, Japan Society for the Promotion of Science (JSPS) KAKENHI Grant No. 21H03409.

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Int. Conf. on Neural Information Process. Syst., 2012, pp. 1097–1105.
[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Conf. on Comput. Vision and Pattern Recognition, 2015, pp. 1–9.
[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Conf. on Comput. Vision and Pattern Recognit., 2016, pp. 770–778.
[4] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: Conf. on Comput. Vision and Pattern Recognit., 2016, pp. 779–788.
[5] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A.W. Senior, K. Kavukcuoglu, WaveNet: A generative model for raw audio, 2016, CoRR arXiv:1609.03499, URL http://arxiv.org/abs/1609.03499.
[6] Z. Chen, X. Huang, End-to-end learning for lane keeping of self-driving cars, in: Intell. Vehicles Symp., 2017, pp. 1856–1860.
[7] F. Codevilla, M. Miiller, A. López, V. Koltun, A. Dosovitskiy, End-to-end driving via conditional imitation learning, in: Int. Conf. on Robotics and Automation, 2018, pp. 1–9.

[8] H. Baomar, P.J. Bentley, Autonomous navigation and landing of large jets using artificial neural networks and learning by imitation, in: Symp. Series on Comput. Intell., 2017, pp. 1–10.
[9] J.S. Denker, Y. leCun, Transforming neural-net output levels to probability distributions, in: Neural Information Process. Syst., 1990, pp. 853–859.
[10] D.J.C. MacKay, A practical Bayesian framework for backpropagation networks, Neural Comput. 4 (3) (1992) 448–472.
[11] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, Y. Wang, VIBNN: Hardware acceleration of Bayesian neural networks, in: Int. Conf. on Architectural Support for Program. Lang. and Operating Syst., 2018, pp. 476–488.
[12] Y. Hirayama, T. Asai, M. Motomura, S. Takamaeda, A hardware-efficient weight sampling circuit for Bayesian neural networks, Int. J. Netw. Comput. 10 (2) (2020) 84–93, URL http://www.ijnc.org/index.php/ijnc/article/view/222.
[13] H. Awano, M. Hashimoto, Bynqnet: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA, in: Design, Automation and Test in Europe, 2020, pp. 1402–1407.
[14] A. Wu, S. Nowozin, E. Meeds, R.E. Turner, J.M. Hernández-Lobato, A.L. Gaunt, Fixing variational Bayes: Deterministic variational inference for Bayesian neural networks, 2018, CoRR arXiv:1810.03958.
[15] M. Haußmann, F.A. Hamprecht, M. Kandemir, Sampling-free variational inference of Bayesian neural networks by variance backpropagation, in: Conf. on Uncertainty in Artif. Intell., 2019.
[16] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, in: Int. Conf. on Machine Learn., 2016, pp. 1050–1059.
[17] Y. Gal, Z. Ghahramani, Bayesian convolutional neural networks with Bernoulli approximate variational inference, in: Int. Conf. on Learn. Representations, 2016.
[18] P. Huang, W.T. Hsu, C. Chiu, T. Wu, M. Sun, Efficient uncertainty estimation for semantic segmentation in videos, in: Eur. Conf. on Comput. Vision, 2018.
[19] G.E.P. Box, M.E. Muller, A note on the generation of random normal deviates, Ann. Math. Stat. 29 (2) (1958) 610–611, http://dx.doi.org/10.1214/aoms/1177706645.
[20] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, J. Cong, FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-hls hybrid templates, in: Int. Symp. on Field-Programmable Custom Comput. Machines, 2017, pp. 152–159.
[21] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, Cudnn: Efficient primitives for deep learning, 2014, CoRR arXiv:1410.0759, URL http://arxiv.org/abs/1410.0759.
[22] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang, A survey of FPGA based neural network accelerator, 2017, CoRR arXiv:1712.08934, arXiv:1712.08934.
[23] Y. LeCun, C. Cortes, C.J. Burges, MNIST handwritten digit database, URL http://yann.lecun.com/exdb/mnist/.
[24] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, (Master's Thesis), Dept. of Comput. Sci., Univ. of Toronto, 2009.
[25] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision? in: Neural Information Process. Syst., 2017, pp. 5574–5584.