Journal of Circuits, Systems, and Computers
© World Scientific Publishing Company

A High-Efficiency FPGA-Based Accelerator for Binarized Neural Network

Submitted for the Special Issue on Design, Technology, and Test of Integrated Circuits and Systems

Peng Guo

Institute of Automation, Chinese Academy of Sciences,
School of Computer and Control Engineering,
University of Chinese Academy of Sciences,
Beijing, 100190, China
iagp1991@163.com

Hong Ma

Institute of Automation, Chinese Academy of Sciences
Beijing, 100190, China
hong.ma@ia.ac.cn

Ruizhi Chen

Institute of Automation, Chinese Academy of Sciences,
Beijing, 100190, China

Donglin Wang

Institute of Automation, Chinese Academy of Sciences,
Beijing, 100190, China

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Although the convolutional neural network (CNN) has exhibited outstanding performance in various applications, the deployment of CNN on embedded and mobile devices is limited by the massive computations and memory footprint. To address these challenges, Courbariaux and Bengio put forward binarized neural network (BNN) which quantizes the weights and activators to ± 1. From the perspective of hardware, BNN can greatly simplify the computation and reduce the storage. In this work, we first present the algorithm optimizations to further binarize the first layer and the padding bits of BNN; then we propose a fully binarized CNN accelerator. With the Shuffle-Compute structure and the memory-aware computation schedule scheme, the proposed design can boost the performance for feature maps of different sizes and make full use of the memory bandwidth. To evaluate our design, we implement the accelerator on the Zynq ZC702 board, and the experiments on the SVHN and Cifar10 datasets show state-of-the-art performance-efficiency and resource-efficiency.

Keywords: CNN; BNN; FPGA; Accelerator.

1. Introduction

In recent years, convolutional neural networks (CNNs) have shown outstanding performance in many fields such as computer vision, speech recognition, and natural language processing [1–3]. However, the huge computation complexity, memory footprint and power consumption of CNNs also bring a tough challenge to the resource-limited embedded platforms. To facilitate the deployment of CNNs, many algorithm-based and hardware-based methods have been proposed.

Considering the noticeable redundancy of CNN models during inference, quantization is taken as a promising approach to compress the CNN models [4–6]. Many works show that low-precision CNNs can achieve comparable accuracy while significantly reduce the memory requirement. In 2016, Courbariaux and Bengio [7] propose one extreme quantization scheme called binarized neural network (BNN) which quantize both the weights and activators to ± 1. Compared to CNN, BNN can not only compress the parameter size by 32x but also convert the fixed-point multiplication operation into a 1-bit XNOR operation. In fact, by representing $+1$ with a set bit and -1 with an unset bit, the multiplication can be performed with an XNOR gate.

With these advantages, the previous work further presents several dedicated BNN accelerators and the results show high performance and power efficiency [8–11]. However, the benefits of BNN have not been fully exploited in their work. Firstly, as the input feature maps (ifmaps) of BNN's first layer commonly are float number, the previous work introduces specific hardware to handle the computation of the first layer, which brings unnecessary loss of throughput and resource efficiency. Secondly, to maintain the accuracy, they need 2-bits padding scheme which also requires more resource overhead. Last but not least, the previous work typically employs different units for the convolutional (Conv) layers and the fully connected (FC) layers.

To address the problems above-mentioned, our work[a] combines the algorithm optimization with the dedicated hardware design. We first present algorithm optimizations to further binarize the convolutional operations across all layers. Then we propose our unified BNN accelerator. We implement it on FPGA and the experiment result shows that it outperforms the state-of-the-art BNN accelerator in power-efficiency and resource-efficiency. The main contributions of our work can be summarized as follows:

- **Fully-binary accelerator architecture:** In this work, we present a scheme to binarize the input of the first layer. Moreover, with negligible accuracy loss, we propose an odd-even padding scheme to represent the padding bits with $+1$ and -1. These optimizations enable our accelerator a fully binary design.
- **Unified computing structure:** By decomposing both the convolution and

[a]This is an extended and revised version of our short paper presented at the 28th International Conference on Field-Programmable Logic and Applications (FPL2018) [12].

dot-product into multiplication and accumulation, we propose a unified Shuffle-Compute structure to support all layers of BNN. Compared with CNN, the binary operation of BNN is much more simplified, which brings the possibility to a higher degree of parallelism. The Shuffle-Compute structure fully exploits this and exhibits sustained high throughput.

- **A memory-aware computation schedule scheme:** Although the memory footprint of BNN has been drastically decreased, some models still cannot be completely loaded into the chip. In order to maximize the performance, we analyze the time budget for the computational and memory operations and introduce a memory-aware computation schedule scheme to parallelize the computation and data loading better.

The rest of the paper is organized as follows. The related work is described in Section 2 and the preliminary is in Section 3. Section 4 details the algorithm optimizations proposed in this paper. Section 5 presents the hardware accelerator. We report our experimental result in Section 6. Finally, the conclusion is presented in Section 7.

2. Related Work

Progress in BNN algorithm Recently, many researchers have demonstrated that the high precision data type is unnecessary for the inference phase, so various quantization strategies are proposed to compress the network. Qiu et al. [13] uses 16-bit data to represent the weight parameters. Gupta et al. [14] presents an 8-bit scheme. Moreover, the extreme low-bit quantization schemes are also proposed. [15] takes advantage of weight sharing to store only a 4-bit index for each weight. Ternary neural networks constrain weight values to 0, +1, or -1 [16], while [17] quantizes the weights to +1 and -1. However, these schemes still keep the native precision of activations and there are also several works seeking to quantize both the weight and activations. For example, [18] represent both weight and activations using 8 bits with negligible accuracy loss. The DoReFa-Net [19] is trained to have 1-bit weights and 2-bit activations. In 2016, Courbariaux and Bengio [7]put forward BNN, which is the most extreme quantization scheme and represents all parameters with one bit. Their training method enables BNN comparable accuracy for the datasets like CIFAR-10 and SVHN. In 2017, [20] optimize the training procedure and significantly improve the accuracy of BNN on ImageNet. Besides, [21] proposes a high-order binarization scheme which achieves more accurate approximation while still preserving the advantage of the binary operation. These advancements further enable BNN a promising approach for the deployment of CNN.

Progress in BNN hardware Meanwhile, there have been many studies on the design of specific BNN hardware. [10] proposes and implements a BNN accelerator on Aria 10 FPGA as well as 14-nm ASIC. Their experiments demonstrate that the dedicated BNN hardware provides superior efficiency over CPU and GPU. But the evaluation of their work only considers the FC layers of BNN, and the size

of test models is less than local memory. [9] develops a BNN framework named FINN, which employs the high-level synthesis to build a hardware module library. According to the specific requirement of performance and power, it can be flexibly configured and compiled into hardware. However, the evaluations are also based on the small models which can be loaded into the local memory and have significant accuracy loss. [8] implements a BNN hardware design on FPGA to accelerate VGG9, which is a classic BNN model for CIFAR-10. It employs independent modules for the first convolutional (Conv) layer, other Conv layers, and FC layers, respectively. [11] proposes a heterogeneous implementation of BNNs on Xeon+FPGA. [22] implements a lightweight YOLOv2 which consists of the BNN and SVR on FPGA for both classification and localization. However, all these works need to tackle with the non-binary input of the first layer, which is against a fully binary accelerator. Also, they do not have an in-depth analysis of the parallel execution of the computation and memory operations for BNN. In this work, with algorithm and hardware optimizations, we fully exploit the advantage of BNN and propose a fully binarized accelerator.

3. Preliminary

3.1. *Basic of BNN*

As we stated before, BNN is an extreme quantization version of CNN. It has a similar topology to CNN which is a kind of directed acyclic graph constructed by stacking multiple Conv layers, Pooling layers, and FC layers. Each layer takes in input feature maps (ifmaps) and weights, performs the corresponding computations and then forwards the output feature maps (ofmaps) to the next layer. By computing layer by layer, BNN can extract the feature information of BNN's input data. A simple BNN model is illustrated in Figure 1.

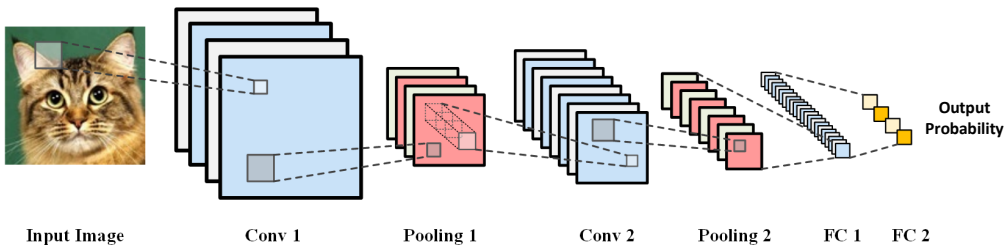

Fig. 1. A simple BNN model. It has two Conv layers, two Pooling layers and two FC layers.

For the Conv layer, suppose the ifmap matrix $\mathbf{A}$ is a 3D matrix, then the calculation can be formulated as:

$$\boldsymbol{O}(z,x,y) = \boldsymbol{sign}(\boldsymbol{batchnorm}(\boldsymbol{A}(k,x,y) \otimes \boldsymbol{W}(z,k,x,y)) \tag{1}$$

$\boldsymbol{O}$ and $\boldsymbol{W}$ represent the matrice of ofmaps and weights; $\otimes$ represents the convolution operation while ***sign*** and ***batchnorm*** are two functions formulated as:

$$\begin{aligned} \boldsymbol{batchnorm}(x) &= \frac{\boldsymbol{x}-\mu}{\sqrt{\alpha^2+\varepsilon}}\gamma+\beta, \\ \boldsymbol{sign}(x) &= \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}. \end{aligned} \tag{2}$$

The ***batchnorm*** function is introduced in [23] while μ, α, ε, γ, and β are all constant numbers decided by the training phase. Compared with the Conv layers of CNN, the main difference of BNN is that the weight matrix $\mathbf{W}$ and the ofmaps $\mathbf{O}$ can only take the value of $+1$ or -1.

The FC layer is also called dense layer, the ifmaps are treated as a 1-D vector and multiplied by the weight matrix, then the intermediate results are normalized with the ***batchnorm*** function and binarized with the ***sign*** function. Similar to the Conv layers, the weight and ofmaps of the FC layers are also binary matrixes.

Compared with CNN, three advantages are introduced by BNN. Firstly, with a set bit representing $+1$ and an unset bit representing -1, one weight parameter can be represented with one bit. Compared with the full precision network, this method can reduce the memory footprint by 32 folds. Secondly, for the ofmaps of each layer are transformed to a binary matrix with the ***sign*** function, the intermediate results also occupy much smaller memory capacity than the normal precise network. Last but not least, from the second layer, the ifmaps of one layer are the ofmaps of the previous layer. So both ifmaps and weights are binary data. As shown in Figure 2, by representing $+1$ with 1, -1 with 0, the multiplications can be transformed into XNOR operations.

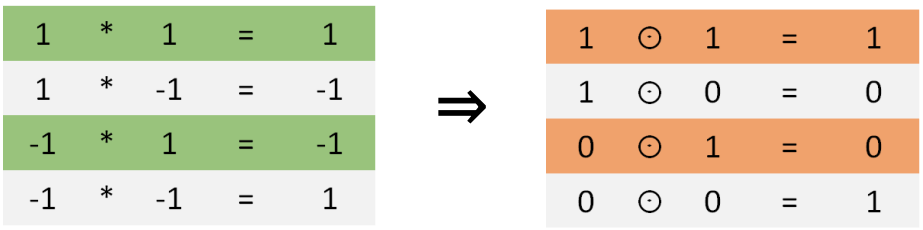

Fig. 2. The multiplication of binary data is transformed into Xnor operation.

3.2. *Trained BNN models*

The key challenge for BNN is how to train the parameters. The traditional CNN models typically take the Back-Propagation (BP) method which updates the pa-

rameters based on the derivative relative to the loss. However, for BNN models, the weights take the discrete value (+1 or −1) and the derived function of the ***sign*** is 0 almost everywhere, so the BP method cannot be applied to BNN directly. In [7], Courbariaux et al. put forward two methods to tackle these problems. Firstly, the weight parameters are taken as binary data during the forward phase while float number during the backward phase, so they can be changed little by little. Secondly, they propose a straight-through estimator to calculate the derivative of the ***sign*** function. In details, suppose the ***sign*** function is $q = Sign(r)$ and the derivative $\frac{\partial C}{\partial q}$ is g_q, then the derivative of C to r is estimated with:

$$\frac{\boldsymbol{\partial C}}{\boldsymbol{\partial r}} = \begin{cases} \boldsymbol{g_q} & |r| \leq 1 \\ 0 & |r| > 0 \end{cases}. \tag{3}$$

With these workarounds, BNN can be trained in a similar way like CNN. In Table 1 we show four typical BNN models: BNN-MNIST, BNN-Cifar10, BNN-SVHN and BNN-SVHN-S, and they will be used in subsequent experiments.[b]

BNN-MNIST is a multilayer perceptron which has three hidden layers of 4096 binary units. MNIST is an image classification dataset which consists of 70K 28x28 gray-scale image representing digits ranging from 0 to 9. The BNN-MNIST model achieves 99.1% accuracy on this benchmark.

BNN-Cifar10 consists of six Conv layers followed by three FC layers. All Conv layers use 3x3 kernels. There is a 2x2 max Pooling layer after the 2nd, 4th, and 6th Conv layers. The BNN-Cifar10 is applied to the Cifar10 dataset which contains 60K 32x32 color images in 10 different classes like airplanes, cars, cats, etc. BNN-Cifar10 achieves 88.5% accuracy for the classification task on this dataset.

The BNN-SVHN has the same topology as BNN-Cifar10, but the number of the units in the Conv layers is halved. The SVHN is also a classification benchmark. It consists of a training set of 604K examples and a test set of 26K 32x32 color images representing digits ranging from 0 to 9. The BNN-SVHN model achieves 97% accuracy. We further halve the number of neurons in the first and second FC layers and retrain it. The simplified model is called BNN-SVHN-S and it significantly reduces the model size with negligible accuracy loss.

4. Hardware-Oriented Algorithm Optimization of BNN

4.1. *Binarize the first layer*

As stated in Section 3.1, one prominent advantage of BNN is that by representing +1 with a set bit and −1 with an unset bit, the multiplication of weights and ifmaps can be executed with XNOR gates. However, one exception is the first layer whose ifmaps are normally fixed-point image data. Most previous work adopts exclusive hardware design to tackle this layer. For example, both [8] and [22] employ an

[b]The first two models are public at https://github.com/MatthieuCourbariaux/BinaryNet.The last two models are trained with the open source method.

Table 1. BNN models for image classification

Model	BNN-MNIST	BNN-CIFAR10	BNN-SVHN	BNN-SVHN-S
Input data	28x28	32x32x3	32x32x3	32x32x3
Weight param.	35.1M	13.4M	6.1M	3.4M
Data Set	MNIST	CIFAR10	SVHN	SVHN
Accuracy	99.1%	88.6%	97.2%	97.0%

independent fixed-point module for the first layer. [10] designs a shareable structure which can be reused as 32 1-bit multipliers or one full-precision multiplier. However, the extra hardware will introduce higher design complexity and power consumption. Moreover, the whole performance is impaired by the first layer's calculations from two aspects. First, fixed-point multiplier consumes much more energy and resources than the XNOR gate. So the number of the fixed-point multipliers is limited in most BNN hardware design. Thus, the performance of the first layer cannot be very high. Second, the input layer normally has three channels, which is often one order of magnitude less than other Conv layers. For the accelerators which map the convolution of multiple input channels to multiple convolvers, the performance of the first layer is limited. For example, the PE in [13] is designed with 64 convolvers and each convolver take charge of one input channel. Thus the utilization for the first layer is only 3/64.

For example, Zhao et al. [8] design a BNN hardware and evaluates it on the BNN-Cifar10 model. For this model, the MAC operations of the first layer account for about 5% of all Conv layers. However, as shown in Table 1 [c], the computation time of the first layer takes about 30% of the total convolution time and the fix-point conv units design also brings about 26% extra area cost.

Table 2. The results of different optimization schemes on [8]

Optimization	Time of Conv1 (ms)	Time of all Conv layers (ms)	Total area (LUT)
Zhao et al. [8]	1.13	3.81	12,456
binarize	4.02	6.70	9,216
binarize + prune	0.5	3.18	9,216
binarize + prune + scale	0.37	**2.36**	12,456

To address this problem, we propose a two-step optimization scheme that con-

[c]The time data is from [8] and the area data is estimated by synthesizing the fix-point and XNOR computing units with Vivado.

sists of binarization and pruning.

Binarization For most BNN network in the computation vision field, the fixed-point input data of most networks is normalized RGB value, which can be scaled into an integer value between -128 and 128. Given an integer a in such interval, it can be essentially represented with the sum of 256 ± 1 as

$$\boldsymbol{a} = \frac{\overbrace{(1+\cdots+1)}^{\boldsymbol{a}+128} + \overbrace{(-1)+\cdots+(-1)}^{128-\boldsymbol{a}}}{2}. \tag{4}$$

Let $\boldsymbol{A}$ be the ifmaps of the first layer, $\boldsymbol{W}$ be the weight matrix. Inspired by the above equation, the convolution of the first layer can be binarized as

$$\boldsymbol{A} \otimes \boldsymbol{W} = \boldsymbol{\alpha} * \sum_{\boldsymbol{i}} \boldsymbol{B_i} \otimes \boldsymbol{W}, \tag{5}$$

where $\boldsymbol{B_i}$ is a binarized matrix and $\boldsymbol{\alpha}$ is a constant. A simplified example is illustrated in Figure 3.

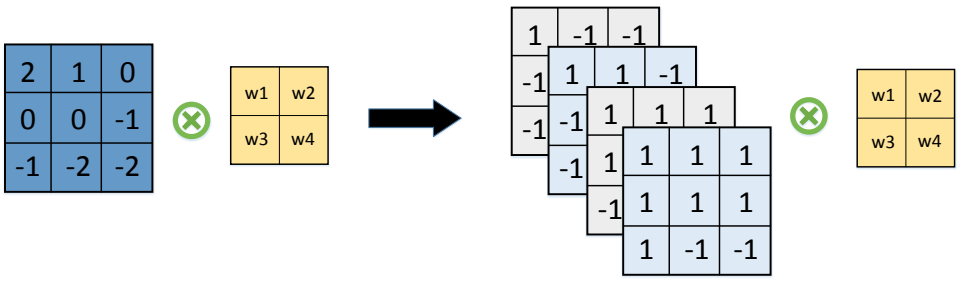

Fig. 3. The original convolution is on 2-bit 3x3 image, then it's expanded into four binary convolutions.

With this binarization scheme, we can employ uniform hardware to process all Conv layers. However, the channel number and XNOR operations of the first layer increase exponentially. As shown in Table 2, we estimate this schemes on [8]. The only binarization optimization option will reduce the area by 26% while increasing the computation time by 75%.

Pruning To resolve the problem of increased calculations, we propose to prune away the low-bits before binarization. If the low N bits of the input fixed-point data are pruned away, the number of the expanded binarized channels will drop by 2^N times. The rationality of the pruning method is as follows: first, the feature information of the input data depends largely on the high-bits of the fixed-point data, and this is why most neural networks are robust to data noise. Second, with binarization on the outputs of each BNN layer, subtle changes of the input data will have little effect on the final accuracy.

We evaluate our method on BNN-Cifar10 and BNN-SVHN-S. As shown in Figure 4, the error rate of BNN-SVHN-S increases from 2.97% to 3.14% when the low 4 bits are pruned away. It is interesting that the error rate of BNN-Cifar10 even decreases from 11.42% to 11.39% when we prune away the low 3 bits. As shown in Table 2, the pruning method can reduce the computation time of the first layer by eight times. Compared with the original scheme, the binarization and pruning scheme reduce the total computation time by 17% with negligible accuracy loss and eliminate the expensive hardware for fixed-point convolutions. Furthermore, by replacing the saved integer computation units with binary computation units, the total time of Conv layers is reduced to 2.36 ms, which is 62% of the original time. A whole two-step optimization scheme of BNN-Cifar10 is illustrated in Figure 5.

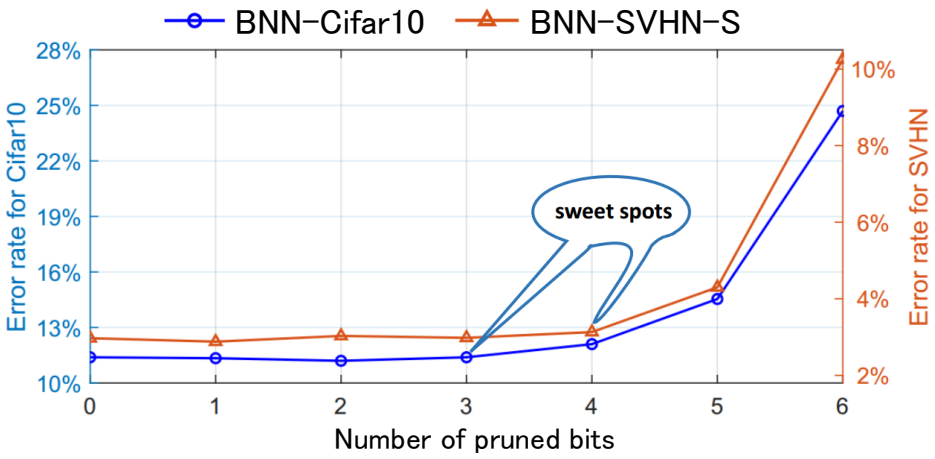

Fig. 4. The accuracy of BNN-Cifar10 and BNN-SVHN when we prune away the low bits.

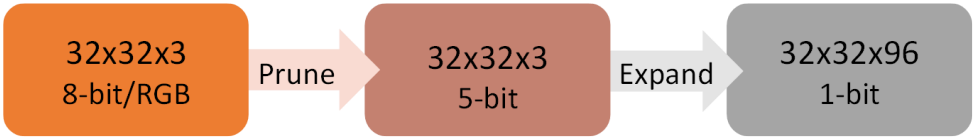

Fig. 5. Apply the pruning and binarization optimizations to the first layer of BNN-Cifar10.

4.2. *Odd-Even Padding*

In traditional CNN, we usually need to pad the ifmaps with 0 in the Conv layers and this is not a big deal. But for BNN, we will need at least two bits to represent +1, −1 and 0 concurrently, which is against our goal to design binary hardware. Previous work [8, 9] pad the ifmaps directly with all +1, however, as shown in Table 3[d], this scheme will increase the error rate significantly.

To neutralize the errors introduced by padding +1, one straightforward scheme is to pad the ifmaps with interleaved (+1,−1), we call this the Odd-Padding. Even though the accuracy is improved through this scheme, it's still worse than the 0-padding. So we take a further step to propose the Even-Padding which pads the ifmaps with interleaved (−1,+1). As shown in Figure 6, we apply the Odd-Padding to the odd channels and Even-Padding to the even channels. Intuitively, this Odd-Even padding can better neutralize the error. In particular, because the RGB input channels share the same weight matrix, this scheme can get the same results as 0 padding on the binarized first layer. As shown in Table 3, The Odd-Even padding has almost the same accuracy as 0-padding.

Table 3. Error rate of different padding methods

BNN Model	0-padding	All +1	Odd-padding	Odd-Even
BNN-SVHN-S	3.14%	3.36%	3.28%	3.15%
BNN-Cifar10	11.39%	13.23%	12.42%	11.25%

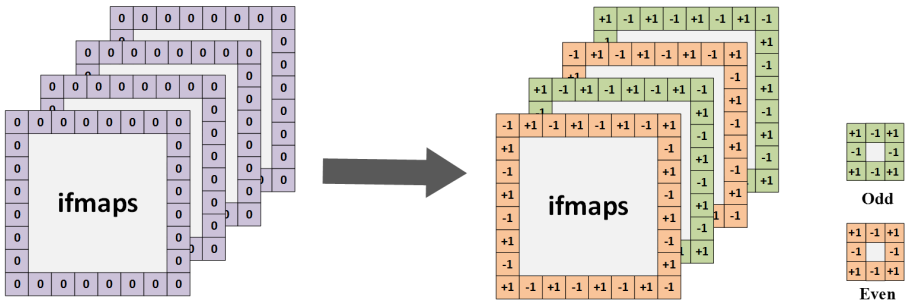

Fig. 6. Even padding and Odd padding are interleaved to replace the zero padding.

[d]The first layer of these models has been binarized with the optimization method introduced in Section 4.1.

5. Accelerator Architecture

5.1. *Design consideration*

With the algorithm optimizations proposed in the previous section, a BNN model is transformed into a fully-binary network. Thus, the memory footprint is greatly reduced and the computations can be performed with bit-wise logical gates. However, an efficient BNN accelerator is far beyond a simplified CNN accelerator with less memory and bitwise computation units. The primary considerations we have while designing the accelerator are listed below.

High-performance and scalable computing system: On the one hand, the computation unit, bandwidth requirement and memory requirement of binary data are much less than that of the full precision data. Under the same resource constraints, we are allowed to provide a higher degree of parallelism. On the other hand, different models are equipped with a different number of layers and channels. The size of the feature maps is also dependent on the dataset and network structure. Moreover, in the practical applications, the hardware resources should be scaled to meet a given classification performance requirement and power requirement. Therefore, the architecture is expected to be scalable and configurable.

Maximizing data reuse: For the computation of BNN, there are three kinds of data reuse. First, all ifmap channels are involved in the computation of one ofmap channel and they are reused for different ofmaps channels. Second, the weight matrix is reused for the convolution on one ifmap channel. Third, the adjacent convolution blocks on one ifmap channel reuse partial ifmap data. On the other hand, the previous work [15] points that the total energy is dominated by the required memory access if there is no data reuse. Therefore, our accelerator should carefully exploit the compute pattern of BNN to maximize the data reuse.

Efficient memory system: Limited to the local memory, most CNN accelerators need to write the intermediate result of each layer to off-chip and load it again later, bringing a lot of energy consumption and performance loss. In contrast, BNN accelerators can keep the intermediate in the local memory. Besides, for the Conv layers, the external bandwidth is normally not fully utilized, making it possible to hide the memory operation. Moreover, we can borrow the bandwidth to prefetch the parameters of FC layers and further improve the whole performance.

5.2. *Overview*

Figure 7 shows the block diagram of the proposed accelerator, which is composed of the processing elements (PEs), the controller and the local memory. With the scheduling of the controller, the DMA loads the input data and weight parameters from DRAM to the local memory, then do the calculations in the PEs. There are Tm PEs and they share the ifmaps data during the computation. For the Conv layers, these PEs take in the same ifmaps channels, perform the computation and generate different channels of ofmaps. For the FC layers, the ofmaps are essentially

1-D vectors and each PE is responsible for one part of the output vector. Except for the last layer, the ofmaps are binary data and stored in the local memory. Besides, the loading of weight parameters is parallelized with the computation. The memory system is detailed in Section 5.5

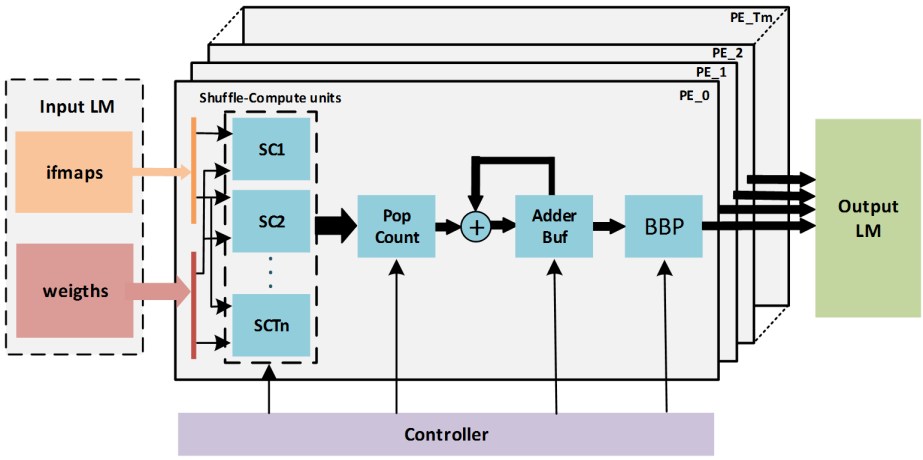

Fig. 7. Architectural block diagram of our BNN accelerator.

The main difference of the Conv layers and FC layers is that the major operation of the former is convolution while that of the latter is vector-matrix multiplication. However, both the operations can be decomposed into multiplication and addition. Besides, the multiplication can be replaced with XNOR operations and the results are still binary data, so the subsequent addition can be performed by counting the number of value +1. Based on this, we propose a uniform structure to support both the Conv layers and the FC layers. As shown in Figure 7, each PE consists of Tn Shuffle-Compute (SC) units, Popcount unit, Adder Buffer, and BBP unit. During the computation, Tn SC units perform the multiplication operations of Tn channels (segments) at one time. The products corresponding to the same position of ofmaps is then summed by the Popcount unit. The partial sum is further accumulated in the Adder Buffer. At last, based on the control signal, the batch norm, binarization and pooling operations are selectively performed in the BBP unit. It is worth noting that the output of the last layer is not binarized. Two techniques introduced in [9] are adopted to optimize the BBP. The first one combines the batch norm and binarizes operation into a direct comparison. The second one swaps the pooling operation and the direct comparison, which will transform the integer comparison into a boolean OR operation.

5.3. *Shuffle-Compute Unit*

The SC unit is responsible for two types of operations: the multiplication in the convolution of the Conv layers and the multiplication in the vector-matrix multiplication of the FC layers. The Conv layers are more complex and computation-intensive, so we first focus on the computation of them. There are two challenges in designing an efficient structure to handle the multiplication of convolution. The first one is the data reuse in one ifmap channel. Supposing the kernel of the convolution is 3x3 and the stride is 1, every two horizontal or vertical adjacent convolution blocks share 6 pixels of the input data. The second one is about the data prefetching. Since the input data and the output data are of the same size when the stride and padding size are 1 (the pooling is performed later), an ideal situation is that the SC unit can load N_i data and output N_i data simultaneously. However, taking a 3x3 convolution as an example, the first two lines of the image and the padding line are needed to calculate the first line of the output image. But a strict prefetch method will introduce the pipeline stall and lower the performance.

In this work, we address the aforementioned problem with the SC structure. As shown in Figure 8 (a), the SC unit consists of the shuffle unit, the ifmaps buffer, the weight buffer and the parallel binary convolver (PBC). Both the ifmaps buffer and the weight buffer are connected to the local memory with a N_i-bit bus. The PBC module takes a 3xN_i structure and each position of the array is essentially an XNOR gate. During the computation, ifmaps buffer takes Ni pixels from the local memory while the shuffle unit transfers 3xNi data into PBC in each cycle. As Figure 8 (b) shows, the data is arranged that each 3x3 block corresponds to one convolution block and the adjacent blocks share 6 ifmaps pixels. The weight buffer loads weight data and broadcasts a 3x3 weight matrix to the Ni convolutional block in PBC. With the XNOR gates in the PBC, the multiplication of Ni convolutions is performed per cycle. Considering that the typical BNN networks only use the 3x3 convolution window size, the SC unit in this work takes a three-line structure. The structure can also be easily extended to support other convolution windows like 5x5 by changing the row number of PBC.

In Figure 9, we further illustrate the calculation flow of the SC unit for the Conv layer. In this example, the ifmaps are 16x16 while the Ni is 64. Each Arabic numeral represents one line. The figure shows three successive cycles. The red frame in the left part indicates the data that is currently read from the input feature map. The red frame in the right part corresponds to the position of the output feature map that is computed in the Shuffle-Compute unit. At each beat, the Shuffle-Compute unit reads 4 rows of input feature map data to the ifmaps buffer. Concurrently the control logic selects appropriate data to the PBC module and then perform the convolution of 64 3x3 blocks. At the first cycle, the ifmaps buffer reads the first four lines of ifmaps' second channel and the computation in PBC is responsible for the last four lines of the ofmaps' first channel. At the second cycle, the ifmaps buffer reads 5-8 lines of ifmaps' second channel while the computation is generating

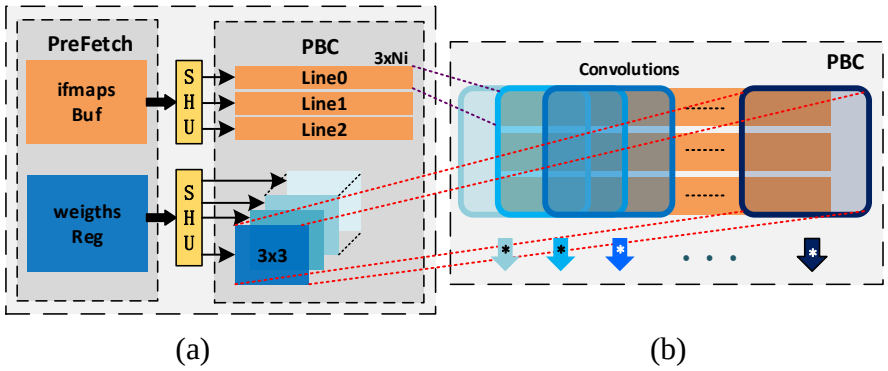

Fig. 8. Block diagram of Shuffle-Compute Unit - (a) overall architecture of SCs; (b) block diagram of PBC with 3 rows and Ni columns.

the first four lines of ofmaps' second channel. At the third cycle, the ifmaps buffer reads the 9-12 lines of the ifmaps while the computation is generating the 5-8 lines of the ofmaps. By pipelining the computation across channels, we succeed to hide the prefetching latency. Besides, when the ifmaps are 32x32, 64x64 or even 128x128, the SC unit can work similarly and still provide sustained peak performance.

For the computation of the FC layers, both the ifmap buffer and the weight buffer take Ni pixels from the local memory and transfer them to the computation line of the PBC directly. Then the dot product of Ni data can be calculated per cycle.

5.4. *Popcount and Adder Buffer*

The Popcount unit is used to count the number of $+1$ in the XNOR results of Tn SC units. Considering the total number of $+1$ and -1 is fixed, we can easily transform this number into to the sum of the multiplication. For FPGA, the Popcount module can be explicitly constructed with LUTs to reduce the resource cost. For an accumulator of 36 bits, our experiment with Xilinx vivado shows that a LUT-based Popcount module needs 32 LUTs while an adder tree needs 42 LUTs, which means 24% resource saving. As we mentioned before, one ofmap channel needs the convolution on all ifmap channels. The partial sum of the convolution or dot product operations are further accumulated in the adder buffer.

5.5. *Memory System*

The parameter size of BNN has been drastically reduced, only 1/32 of the full precision CNN. Even some large ifmaps, like 64x64x256, can be kept in most embedded devices easily. By using the ping-pong structure with two memory banks, we can keep the ifmaps/ofmaps always on the chip, which greatly minimizes the off-chip memory access.

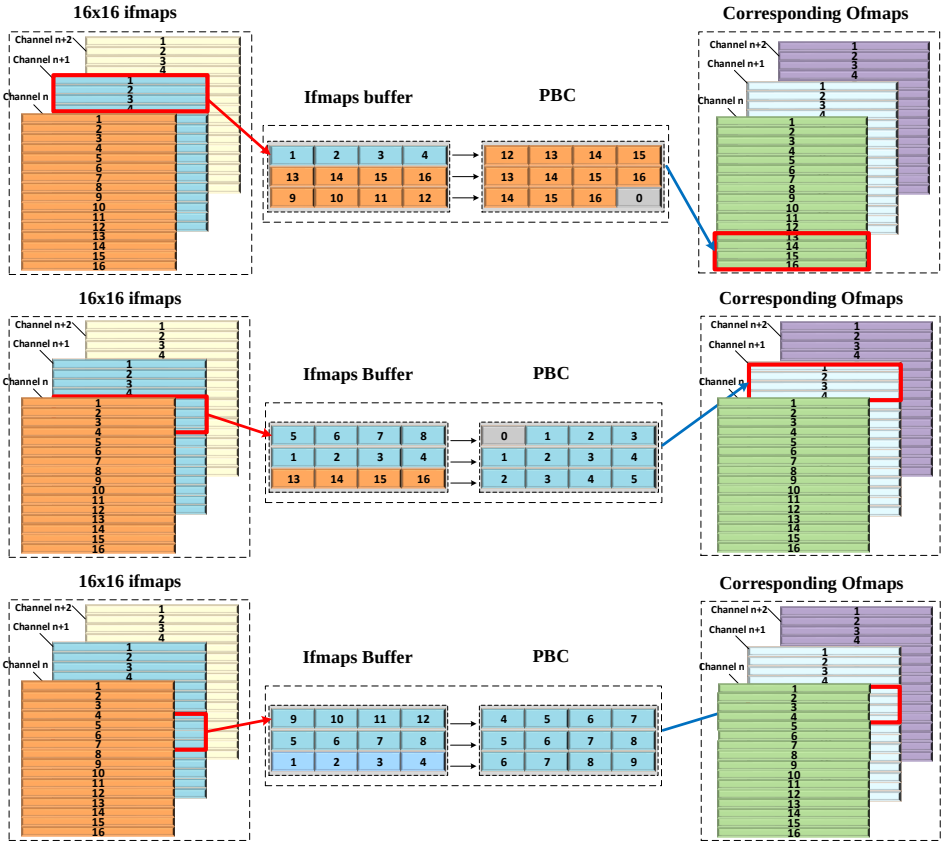

Fig. 9. The data flow of the Shuffle-Compute unit during successive three cycles. Both ifmap and ofmap are 16x16 and the Ni is 64. Each number represents one line of the feature maps and 0 represents the padding line. The left part represents the input feature map while the right part represents the output feature map. The middle represents the Shuffle-Compute unit.

However, the weight size may still exceed the capacity of some typical FPGA devices, like Xilinx ZC702, which has a maximum of 0.5 MB on-chip memory, while the weight parameters of binarized AlexNet and VGG16 are 7.6 MB and 17 MB, respectively. The design in [8] serially executes the computation and memory operations. It first loads the weight from off-chip and then does the computation. The Conv layer is computation bounded and the memory access time can be hidden theoretically. A straightforward optimization is to embed two weight banks and parallel the computations of layer M and the memory operations of layer $M+1$. However, the parallelism on the granularity of each layer cannot be effective for various BNN networks. For example, some layer's weights may exceed the capacity of the weight banks. Besides, the computation amounts vary across different layers, which brings low efficiency with the coarse parallelization scheme.

As shown in Figure 10, we divide the Conv operations of each layer into several phases. The rules of partition strategy is as follows: (1) Each phase belongs to one Conv layer or one FC layer; (2) The weights of every phase need to be less than the capacity of the memory bank; (3) For phase N and phase $N+1$, the computation time of phase N must be longer than the DRAM time of phase $N+1$. With such a scheduling scheme, the DRAM time for the Conv layers is completely hidden and we can also parallel the computation and memory operations for the FC layers, leading to better bandwidth and performance efficiency.

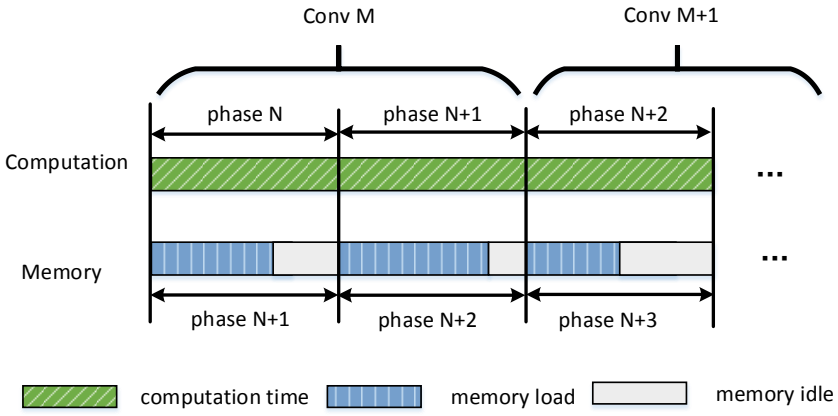

Fig. 10. The Conv layers are divided into phases and we parallel the computations of phase N and the DRAM operations of phase $N+1$.

6. Experiment

6.1. *Experimental Setup*

To evaluate our design, we implement our design on Xilinx Zynq ZC702 evaluation board. The chip on this board integrates a Xilinx Artix-7 FPGA and a dual-core ARM Cortex-A9 MPCore. The accelerator is mapped to the FPGA and connected with DRAM through a 64-bit AXI4 HP port. All synthesis results are obtained from Xilinx Vivado 2016.4. In our experiments, the accelerator core is implemented at 143 MHz while the HP port is at 250 MHz.

We fully binarized the BNN-Cifar10 and BNN-SVHN-S as our experimental networks, consisting of 6 Conv layers and 3 FC layers. The binarized BNN-Cifar10 has a total of 1386 Giga computation operations and 1.7 MB weights, which can achieve 88.6% accuracy on Cifar10. The binarized BNN-SVHN-S has only 346 Giga operations and 0.45 MB weights, achieving 96.9% accuracy on SVHN dataset.Besides, the input images are binarized and pruned in the CPU part of the board.

6.2. *Configuration*

Our design can be scaled with different configurations of Tm, Tn, and Ni. As 8x8 is the minimum ifmap size of both the BNN-Cifar10 and BNN-SVHN-S, we use $Ni = 64$. The resource utilization and theoretical performance of our architecture under different Tm and Tn are shown in Table 4. The peak performance is proportional to $Tm \times Tn$. As the SC units in same PE share the adder buffer and BBP module, increasing Tn would be more resource efficient than increasing Tm. However, Tn is bandwidth-constrained and also restricted by the computations of FC layers. First, as different SC units take in different ifmap channels, increasing Tn brings growing need of the local bandwidth. Second, both BNN-Cifar10 and BNN-SVHN-S have up to 1024 neurons in the FC layers, so increasing Tn to larger than $1024/64 = 16$ will not improve the performance for the FC layers. Limited to the capacity of ZC702 in our experiment, we finally use $Tm = 1, Tn = 16$ in the following experiment.

Table 4. Resource utilization and theoretical peak performance under different configurations

Tm	Tn	LUT	FF	Block RAMs	Peak Performance (GOPS)
1	2	13,273	17,531	103	331
1	4	16,139	21,131	103	663
1	8	19,929	23,876	103	1,327
2	4	27,305	29,143	103	1,327
1	**16**	**29,629**	**31,763**	**103**	**2,654**
2	8	34,866	36,915	103	2,654
Dev.		53,200	106,400	140	-

6.3. *Comparison*

As shown in Table 4, we first compare our design with other two general platforms: the Intel 6700K CPU and the NVIDIA GTX1070 GPU. Then, we compare our design with two FPGA-based BNN accelerators: FINN [9] and the accelerator proposed by Zhao et al. [8]. We use GOPS per kLUT and GOPS per watt to represent the resource efficiency and power efficiency. In Table 6, we compare our work with several FPGA based CNN accelerators.

CPU/GPU Both CPU and GPU are evaluated with theano on PC. The power of CPU is detected with the *turbostat* command while the power of GPU is reported with the $nvidia - smi$ command. Compared with CPU, our design achieves about 32x better performance and 559x higher power efficiency on SVHN. On Cifar10, we achieve 5x better performance and 94x higher power efficiency. The large variance between the two models is due to the capacity of on-chip memory, in which the

Table 5. Comparison with other BNN platforms for SVHN and Cifar10

Platf.	Dataset	Accu.	kLUTs	GOPS	Power (W)	GOPS/ kLUT	GOPS/ W
CPU	SVHN	97.1%	-	69	55.2	-	1.25
GPU	SVHN	97.1%	-	2708	133	-	9.25
FINN	SVHN	94.9%	46.2	2465	3.6	53.3	684.7
Ours	SVHN	96.9%	29.6	2236	3.2	75	699
CPU	Cifar10	88.58%	-	135	58.1	-	2.32
GPU	Cifar10	88.58%	-	3380	147	-	24.7
FINN	Cifar10	80.0%	46.2	2465	3.6	53.3	684.7
Zhao	Cifar10	88.54%	46.9	208	4.7	4.43	44.2
Ours	Cifar10	88.61%	29.6	722	3.3	24	219

Cifar10 model exceeds the capacity and the performance is significantly limited by the bandwidth. Even though the performance of GPU outperforms our design, we achieve 75x and 8.9x power efficiency on SVHN and Cifar10, respectively.

FINN The results of FINN [9] are evaluated on a tiny BNN network. For SVHN, FINN shows 10% better performance, but our design is 1.4x resource efficient. For Cifar10, although their test model also employs six Conv layers and three FC layers, the channel number of its Conv layers and neuron number of its FC layers are reduced by half than BNN-Cifar10. Besides, it does not pad the ifmaps in the Conv layers, so the ifmaps will get smaller even if there are no pooling operations. With these modifications, the parameters of the network are only 0.19MB and the operand can be represented with 1 bit, thus the model can be totally stored in the on-chip memory and the hardware design can take a binary computation unit. Overall, FINN presents better resource efficiency and power efficiency. However, these modifications also bring significant accuracy loss for the Cifar10 dataset. As shown in Table 5, it achieves 80.0% accuracy while our model achieves 88.6%. So the comparison in performance is unfair to some extent.

Zhao et al. The design in [8] is implemented with the same FPGA platform and the same BNN-Cifar10 model as our design. Both designs are implemented under 143 MHz and show similar accuracy. So it is really suitable as the reference design.

As shown in Table 5, [8] takes 5.94 ms while our work only needs 1.92 ms per image, which is a 3.1x speedup. Besides, our work achieves 5.4x resource efficiency and 4.9x power efficiency, respectively. We detail the comparison in Figure 11. For the first layer, as described in Section 4.1, [8] takes a dedicated module for the computation of the first Conv layer, and they can perform 3 3x3 convolutions per cycle. In contrast, we expand the first layer from 3 RGB channels to 96 binary channels and employ the uniform SC units to tackle with the computation. Under

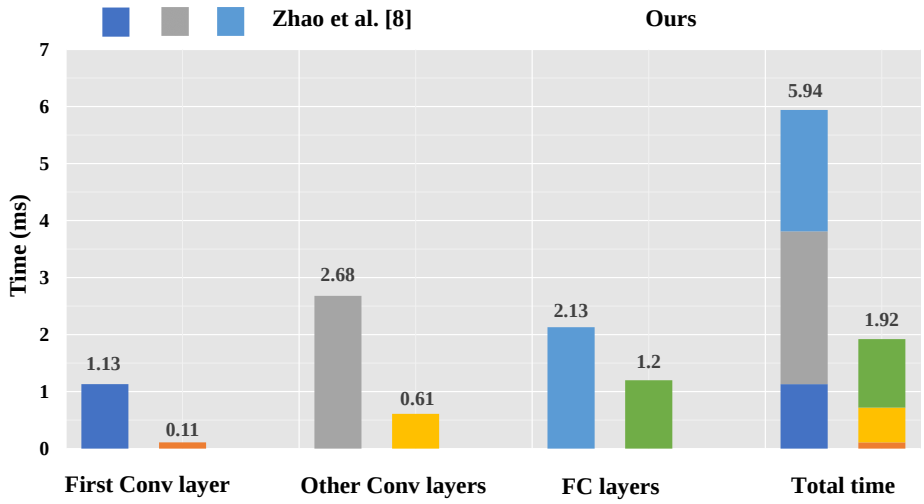

Fig. 11. Detailed timing comparison with [8]

the selected configuration, our design can perform 16x64 3x3 convolutions per cycle, thus can bring about 341x speedup for the first Conv layer theoretically. Eventually, subject to data handling and other operations, our design takes 0.11ms to compute the first layer while [8] takes 1.13ms. For the other Conv layers, our design takes 0.61ms while the reference design needs 2.68ms. On the one hand, the reference design can perform 8x64 3x3 convolutions per cycle. Due to the unified binary PE structure, our design is more resource efficient and can deliver twice the peak performance. First, as indicated in Table 2, by removing the dedicated module for the first layer, our design leaves more resources for the PE units. In addition, the fully binary structure and reusing SC units for the FC layers also contribute to the high resource efficiency. As shown in Table 6, we present a PE with 2-bit input and a PE with separated FC computation unit to quantitively analyze the benefits. We can find that the 2-bit version takes 1.8x more LUTs than our design while the separated FC unit version takes 1.07x. On the other hand, the memory system enables our design hiding the time for loading weights. For the FC layers, our design takes 1.2ms while [8] takes 2.13ms. The computation of the FC layers is memory bounded and we own the improvement to the memory system. Actually, our design prefetches the partial weights during the computation of the Conv5 layer and we also parallel the computational and the memory operations.

FPGA based CNN accelerators As a comprehensive evaluation, we compare our BNN accelerator with other FPGA-based CNN accelerators in Table 6. Our binarized accelerator achieves significant improvements in performance, resource efficiency, and power efficiency, which fully demonstrates the advantage of the binarized network. It may not be unfair to directly compare CNN accelerator and

Table 6. The resource cost of different type of PE

Type of PE	**Uniform 1-bit**	**2-bit version**	**1-bit+separated FC**
Resource (LUT)	24,139	43,896	25,717

BNN accelerator. However, it is currently the standard practice for hardware accelerator studies to compare the reduced and full-precision implementations. Besides, BNN has exhibited comparable accuracy with CNN for middle-scale applications and the accuracy is still improving [20, 21].

Table 7. Comparison with FPGA based CNN accelerators

Design	**[24]**	**[13]**	**[25]**	**Ours**
Device	Virtex VX485t	Zynq XC7Z045	Virtex7 VX690T	Zynq XC7Z020
Clock(MHz)	100	150	120	143
Data Precision	32-bit float	16-bit	8-16bit	1-bit
kLUT	186	182.6	115	34.9
DSP	2240	780	1436	0
Power(W)	18.6	9.63	24.8	3.3
GOPS	61.62	137.0	222.1	722
GOPS/kLUT	0.33	0.75	1.93	24
GOPS/Watt	3.31	14.3	8.96	219

7. Conclusion

In this paper, we propose a fully binarized neural network accelerator. With the hardware-oriented algorithm optimizations, we binarize the convolution across all the layers and construct a unified hardware design for BNNs. The SC units are proposed to provide sustained peak performance for the convolutions. And the memory system further improves the efficiency and performance of our design. We implement our design with Xilinx ZC702 and fully evaluate its performance on Cifar10 and SVHN. On Cifar10, it achieves 722 GOPS overall performance, 24 GPOS/KLUT resource efficiency, and 118 GOPS/watt power efficiency, which has 3.1x, 5.4x, and 4.9x improvements of the state-of-art works.

References

1. X. Wang, A. Shrivastava, and A. Gupta, "A-fast-rcnn: Hard positive generation via adversary for object detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
2. G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks." in *ICASSP*, vol. 14. Citeseer, 2014, pp. 4087–4091.
3. B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Advances in neural information processing systems*, 2014, pp. 2042–2050.
4. S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
5. A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.
6. V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
7. M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.
8. R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 15–24.
9. Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.
10. E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: comparison of fpga, cpu, gpu, and asic," in *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 2016, pp. 77–84.
11. D. J. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "High performance binary neural networks on the xeon+ fpga™ platform," in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*. IEEE, 2017, pp. 1–4.
12. G. Peng, M. Hong, C. Ruizhi, L. Pin, X. Shaolin, and W. Donglin, "Fbna: A fully binarized neural network accelerator," 2018.
13. J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.
14. S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.
15. S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016, pp. 243–254.

16. F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.
17. M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.
18. D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.
19. S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.
20. W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *AAAI*, 2017, pp. 2625–2631.
21. Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao, "Performance guaranteed network acceleration via high-order residual quantization," *arXiv preprint arXiv:1708.08687*, 2017.
22. H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight yolov2: A binarized cnn with a parallel support vector regression for an fpga," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018, pp. 31–40.
23. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
24. C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.
25. Z. Liu, Y. Dou, J. Jiang, and J. Xu, "Automatic code generation of convolutional neural networks in fpga implementation," in *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, 2016, pp. 61–68.