An Energy-Efficient Architecture for Binary Weight Convolutional Neural Networks

Yizhi Wang, Jun Lin, *Member, IEEE*, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*— Binary weight convolutional neural networks (BCNNs) can achieve near state-of-the-art classification accuracy and have far less computation complexity compared with traditional CNNs using high-precision weights. Due to their binary weights, BCNNs are well suited for vision-based Internet-of-Things systems being sensitive to power consumption. BCNNs make it possible to achieve very high throughput with moderate power dissipation. In this paper, an energy-efficient architecture for BCNNs is proposed. It fully exploits the binary weights and other hardware-friendly characteristics of BCNNs. A judicious processing schedule is proposed so that off-chip I/O access is minimized and activations are maximally reused. To significantly reduce the critical path delay, we introduce optimized compressor trees and approximate binary multipliers with two novel compensation schemes. The latter is able to save significant hardware resource, and almost no computation accuracy is compromised. Taking advantage of error resiliency of BCNNs, an innovative approximate adder is developed, which significantly reduces the silicon area and data path delay. Thorough error analysis and extensive experimental results on several data sets show that the approximate adders in the data path cause negligible accuracy loss. Moreover, algorithmic transformations for certain layers of BCNNs and a memory-efficient quantization scheme are incorporated to further reduce the energy cost and on-chip storage requirement. Finally, the proposed BCNN hardware architecture is implemented with the SMIC 130-nm technology. The postlayout results demonstrate that our design can achieve an energy efficiency over 2.0TOp/s/W when scaled to 65 nm, which is more than two times better than the prior art.**

***Index Terms*— Approximate computing, binary weight convolutional neural network (BCNN) architecture, convolutional neural network (CNN), deep learning, energy-efficient design, signal processing, VLSI architecture.**

I. Introduction

CONVOLUTIONAL neural networks (CNNs) have achieved great success in several fields such as image classification [1], motion recognition [2], and speech recognition [3], as well as many other big-data analysis tasks. As various kinds of CNNs presenting such fantastic performance, models also become deeper (e.g., models proposed in [4]–[6] have depths of 19, 22, and more than 100 layers, respectively), so as to increase the classification accuracy, which leads to extremely high computation complexity. The most widely used approach to accelerate deep learning algorithm is through GPUs, which can reach very high throughput of up to 6TOp/s. However, the side effect is that it costs about 250 W [7], which means that it is impossible to apply complicated algorithms in small embedded systems via high-performance GPUs because of limited energy budget.

To reach the goal of both energy efficiency and high performance when implementing CNNs, both algorithmic and hardware architectural solutions have been studied. One approach is to implement CNNs in hardware like ASICs and field-programmable gate arrays (FPGAs). Chen *et al*. [8] explored a row-stationary dataflow on a spatial architecture combined with network-on-chip technique to perform inter-processing element communication. The architecture proposed in [9] is a completely on-chip processing system for CNNs, which can store an entire CNN model on chip and eliminate all DRAM accesses for weights. This architecture makes it possible to be integrated with a CMOS or charge-coupled device sensor. However, the benchmarks it used are all less than four convolutional layers, and for very deep CNN models like visual geometry group (VGG)-16 [4] (13 convolutional layers), it is not applicable due to the limited on-chip SRAM capacity. A scalable CNN accelerator called Origami was proposed in [10]. The remarkable achievement of Origami is that it can limit the external memory bandwidth to just 525-MB/s full-duplex. The resistive random access memory-based CNN accelerator [11] has better energy efficiency compared with FPGA-based implementations. However, memristors suffer from various nonideal effects and may decrease accuracy significantly. Though these architectures are more energy efficient compared with GPUs, still they cannot satisfy the requirements of embedded Internet-of-Things (IoT) systems. What is more, most of these works cannot fit very deep CNN models well.

Another approach is to modify the CNN models in order to make them applicable in power-sensitive and storage-limited devices. Various kinds of quantization and model compression schemes have been proposed to reduce the model size [12]–[15]. Directly quantizing the model will bring accuracy loss, and finding the optimal number of quantization bits is time-consuming [12]. Deep compression [13] method can reduce the size of AlexNet [1] by 35 times with almost no accuracy loss. However, extra workloads of pruning, trained quantization, and Huffman coding make it more complicated in both training and testing. Besides, it also leads to extra computation complexity due to data restoration.

Another promising method that can overcome these drawbacks is to train weights and even activations into binary values. Binary weight CNN (BCNN) was firstly investigated in [16], where a simplified neural network called

Manuscript received January 18, 2017; revised June 3, 2017, July 30, 2017, and September 9, 2017; accepted October 17, 2017. This work was supported in part by the National Natural Science Foundation of China under Grant 61774082 and Grant 61604068 and in part by the Fundamental Research Funds for the Central Universities under Grant 021014380065. *(Corresponding authors: Jun Lin; Zhongfeng Wang.)*

The authors are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210008, China (e-mail: magicwyzh@smail.nju.edu.cn; jlin@nju.edu.cn; zfwang@nju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2767624

BinaryConnect was proposed. In BinaryConnect, weights are trained into $+1$ and -1, such that no multiplications are needed in the convolutional and fully connected layers. Later in a further work, the same research group proposed the binarized neural network (BNN) [17], which binarizes not only weights but also activations into binary values. Thus, multiplications and additions in convolutional and fully connected layers can be transformed to bitwise XNOR operations.

However, both BinaryConnect and BNN work well on only small data sets such as Mixed National Institute of Standards and Technology (MNIST), Canadian Institute for Advanced Research (CIFAR)-10, and Street View House Numbers (SVHN) [18], because straightforward binarization methods will cause significant error. Binary weight network (BWN) and XNOR-Net were developed in [18]. BWN works nearly the same as the BinaryConnect but employs some extra scaling factors to minimize the mean-squared difference between outputs of a BWN and that of the original full-precision one. XNOR-Net binarizes activations as well as weights just as BNN does, but it brings in one more compensation scaling term in each output feature map of a convolutional layer in order to limit the error in binarized activations. On very large data set like ImageNet (1.28 million images with up to 1000 categories), BWN can still keep a high classification accuracy, which transcends BinaryConnect and BNN significantly (17% and 25%, respectively). Compared with the corresponding full precision CNN model, BWN has only about 3% degradation in both top-1 and top-5 classification accuracy, which is very attractive. As for XNOR-Net, though it can outperforms BinaryConnect and BNN on very large data sets too, it still suffers from about 12% accuracy loss in comparison with its full precision version.

Several hardware architectures for BCNNs have been presented. In [19], an efficient hardware architecture of the BinaryConnect algorithm, which is called YodaNN, was proposed. The YodaNN requires relatively small area and energy consumption since complicated floating multiplications and accumulations (MAC) operations are replaced with simpler complement operations and multiplexers. In terms of energy efficiency, the implementation results in [19] demonstrate that BCNN is better than full precision CNNs. However, still there exist some shortcomings in YodaNN since it needs to exchange data with off-chip DRAM for several times. This results in a waste of large amount of energy because of heavy I/O access. Besides, a framework called FINN [20] was presented for building FPGA accelerators for BNN, enabling efficient mapping of model to hardware. Because FPGA implementation consumes more energy than ASIC, still it is infeasible for embedded IoT systems with limited energy budget.

In this paper, a high-performance energy-efficient BCNN architecture is proposed, which is suited for low-power embedded systems. It is well optimized for BCNN's binary weights, resulting in less off-chip I/O access and reduced power dissipation. This architecture can support very deep binarized CNN models and is compatible with both BinaryConnect and BWN algorithms. The main contributions of this paper can be summarized as follows.

1) We propose an architecture and the corresponding processing schedule optimized for very deep BCNN models. Most parts of the design consideration are toward low energy cost and utmost data reuse.
2) Algorithmic transformation and microarchitectural level optimization, including compressor tree, negative skipping, and earlier pooling, are proposed to reduce the delay in the data path and energy consumption.
3) We introduce approximate computing in binary multiplications (the input activations only multiply $+1$ or -1) and two compensation schemes to our design, which can significantly reduce the number of adders while causing little or no accuracy loss. Besides, BCNN's robust property to the noise brought by imprecise adders is also investigated. Data path of our architecture is equipped with dedicated approximate adders, contributing to less area and delay.
4) The memory efficient quantization scheme for BCNNs that can reduce the size of memories for storage of intermediate data is discussed. Besides, experiments on several data sets are presented to prove the general applicability of this scheme.
5) The proposed architecture is implemented and evaluated. Comparisons with prior arts are also shown in this paper.

The rest of this paper are organized as follows. Section II introduces related fundamental concepts of CNNs and BCNNs. Some optimizations for algorithmic strength reduction and hardware design are presented in Section III. In Section IV, the architecture of the proposed hardware is elaborated, showing how to optimize both microarchitecture and processing schedule for BCNNs. Section V presents the implementation results and compares this paper with the state-of-the-art BCNN architectures. Finally, conclusions are drawn in Section VI.

II. Background

A. Convolutional Neural Network

A typical CNN is mainly composed of several basic stages, each of which is a concatenation of convolution, normalization, activation (nonlinear), and pooling layers. Modern deep CNNs mostly consist of many basic stages to work as feature extractors. Besides, there are a series of fully connected layers following the the last stage to finish the final classification task. Since convolutional layers are more computation-intensive than fully connected layers, we mainly focus on convolutional layers in this paper.

Let $\ell = 1, 2, \ldots, n$ be the index of a basic stage. A typical basic stage is

$$\mathbf{y}^{(\ell)} = \text{conv}(\mathbf{x}^{(\ell)}, \mathbf{k}^{(\ell)}) + \mathbf{b}^{(\ell)} \tag{1}$$

$$\mathbf{v}^{(\ell)} = \text{BatchNorm}(\mathbf{y}^{(\ell)}) \tag{2}$$

$$\mathbf{x}^{(\ell+1)} = \text{MaxPool}(\text{act}(\mathbf{v}^{(\ell)})) \tag{3}$$

where $\mathbf{x}$ is the input of the ℓth stage, and $\mathbf{x}^{(1)}$ is the input image. $\mathbf{b}$ denotes the bias term of the corresponding convolutional layer. $\mathbf{y}$ represents the output feature maps of a convolutional layer, and each output feature map is an output channel. A feature map is a set of neurons (also called

activations) with a size of $h \times w$, and the output feature maps of one stage are the input feature maps of the next stage. The convolution operation in (1) computes

$$y_o^{(\ell)}(j,i) = b_o^{(\ell)} + \sum_{c \in C_{\text{in}}^{(\ell)}} \sum_{(m,t) \in S^{(\ell)}} k_{o,c}^{(\ell)}(m,t) x_c^{(\ell)}(j+m, i+t) \tag{4}$$

where o and c are the indices of the output and input channels, respectively. y_o denotes an output neuron of channel o, and x_c is an input neuron of channel c. y_o is calculated within a small region $S^{(\ell)}$ of size $w_{\text{kernel}} \times h_{\text{kernel}}$ (e.g., 3×3) by convolution weight k. It is common to refer to the sets of weights with a size of $w_{\text{kernel}} \times h_{\text{kernel}}$ as a kernel, that is convolved with the input. For each output channel, there can be an optional bias term b_o added to each of the neuron.

Batch normalization layers [21] in (2) can bring in many advantages like faster convergence in training and preventing the model from overfitting. With batch normalization, the output of each convolutional layer can be normalized to reduce internal covariate shift. It is essential to keep a batch normalization layer after each convolutional layer; otherwise, a very deep CNN or BCNN model is hard to converge in finite time. The simple batch normalization operation without trainable parameters can be described as a linear transformation applied to each output neuron of the convolutional layer given by

$$v_o^{(\ell)}(j,i) = (y_o^{(\ell)}(j,i) - \mu_o^{(\ell)}) \times \frac{1}{\sigma_o^{(\ell)}} \tag{5}$$

where μ_o and σ_o are statistical data obtained during training time.

In most of the state-of-the-art CNN models, rectified linear unit (ReLU) layers and max pooling layers are mostly used within a basic stage. The ReLU activation function applies $\text{act}_{\text{ReLU}}(x) = \max(0, x)$ to every input neuron from the normalization layer. The max pooling is a down-sampling operation that finds the maximum from a small neighborhood for each channel, often on 2×2 region and with a stride of 2×2. Thus

$$\begin{aligned} x_o^{(\ell)}(j,i) &= \max\big(v_o^{(\ell)}(2j, 2i), \quad v_o^{(\ell)}(2j, 2i+1) \\ &\qquad \times v_o^{(\ell)}(2j+1, 2i), \quad v_o^{(\ell)}(2j+1, 2i+1)\big) \end{aligned} \tag{6}$$

where $v_o^{(\ell)}$ and $x_o^{(\ell)}$ are the outputs of ReLU and max pooling layers, respectively. ReLU layer can make the inputs of the next stage sparse, and pooling layer can significantly reduce the dimension of the next stage's inputs.

B. Binary Weight CNN

Current BWCNN architectures include BinaryConnect [16], BNN [17], BWN, and XNOR-Net [18]. In this section, we mainly focus on the BinaryConnect and BWN, which only binarize their weights while keeping their activations with high precision.

A BCNN is similar to an original CNN model except the binary weights and some compensations. In BinaryConnect, all weights are trained as

$$k_b = \begin{cases} 1, & \text{if } k_{\text{fp}} > 0 \\ -1 & \text{if } k_{\text{fp}} < 0 \end{cases}. \tag{7}$$

$k_{o,c}$ in (4) is constrained to $+1$ or -1 according to the sign of its original full precision weight k_{fp}, which is preserved during training.

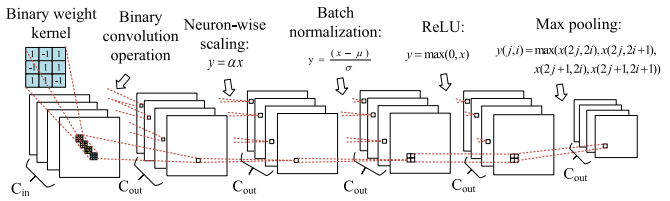


Fig. 1. Basic stage of BCNN.

BWN works nearly the same as BinaryConnect except that it keeps an extra scaling factor α for all output channels in a convolutional layer. If we force $\alpha = 1$, BWN degenerates to BinaryConnect. The output of BWN's convolutional layer is formulated as

$$y_{o,\text{bwn}}^{(\ell)}(j,i) = \alpha_o^{(\ell)} \times y_{o,\text{bc}}^{(\ell)}(j,i) \tag{8}$$

$$\alpha_o^{(\ell)} = \frac{\|\mathbf{W}_{o,\text{fp}}\|_{\ell 1}}{n} \tag{9}$$

where y_{bc} denotes the BinaryConnect output neuron and y_{bwn} represents the BWN output. $\mathbf{W}_{o,\text{fp}}$ is the full precision weight matrix for output channel o, which is kept during training. n is the number of weights in the weight matrix. Note that here the bias term is ignored for simplicity, and during the inference time only $\alpha_o^{(\ell)}$ needs to be stored on-chip. Fig. 1 illustrates a basic stage of a BCNN.

III. Algorithmic Optimization for BCNNs

A. Approximate Binary Multiplication With Compensation

Each weight of a BCNN is binarized to $+1$ and -1, which means all multiplications in convolutional layers are eliminated and there are only summations and subtractions left in convolution. In this paper, we choose to use two's complement representation to represent data rather than representing data in the sign-magnitude format. This is because a sign-magnitude representation system needs extra data conversion units or specific subtracter to add negative values.

Since only binary weights are used in BCNNs, original multiplications can be replaced by binary multiplications with two's complement operations if the activations multiply -1. However, a two's complement operation still consumes an extra add operation to add 1 to the one's complement of the input data. In order to further reduce the algorithmic complexity, we propose to use one's complement instead of two's complement. However, this approximation results in considerable calculation error during accumulation and causes an accuracy loss of up to 15% (tested on CIFAR-10 data set with VGG-16 model).

Taking the error source into consideration, it can be found that the relationship between each output neuron with error x^*

and its correct value x satisfies

$$x^* = x - n \tag{10}$$

where n denotes the number of related weights trained to -1.

Here, we propose two efficient compensation schemes. The first one is to approximately compensate the value of 0.5 $C_{\text{in}}w_{\text{kernel}}h_{\text{kernel}}$ to each output neuron just like a bias term. The main motivation is that statistically the possibilities of a w as above, the experimental result of this scheme shows only 0.71% accuracy loss and there is no extra resource cost at all. Another one is to precisely ceight being $+1$ and -1 are roughly equal. With other conditions being the sameom-pensate n, which can be obtained in advance, to each output neuron. This scheme needs to save extra C_{out} parameters for a convolutional layer, but it can achieve no accuracy loss while eliminating a great number of adders as mentioned above. Choosing either of the two approaches is an issue of tradeoff between accuracy and on-chip storage. In the implementation demonstrated in Section V, the second scheme is chosen.

B. Earlier Pooling

The max pooling layer selects the maximum of every four input neurons (2×2 pooling) in each window of a feature map. If the computation flow is the same as (1), (2), and (3), all four input data should be processed before pooling. It costs extra energy because only the maximum one is needed. To eliminate the unnecessary computation, an earlier pooling scheme is proposed. Scaling and batch normalization operations work together as a linear transformation for each neuron, where $\alpha > 0$ and $\sigma > 0$. Thus, these two operations do not change the relative magnitude between inputs of the following max pooling layer. We propose to perform convolution-MaxPooling-scaling-BatchNorm-ReLU when max pooling layer exists. As a result, only $1/4$ of the neurons are fed to the following layers after convolution and max pooling.

C. Complexity Reduction for Linear Transformation

The linear transformation of each neuron is formulated as

$$y_{lt} = (y_{\text{conv}} \times \alpha - \mu) \times \left(\frac{1}{\sigma}\right) \tag{11}$$

where y_{conv} is the output of a convolutional layer, α is the scaling factor of BWN (for BinaryConnect approach, $\alpha = 1$), and μ and σ are the running mean and standard deviation in batch normalization layers, respectively. Two multiplications will be needed if it is calculated in the direct way. In order to reduce the multiplication complexity, we reformulate it to

$$y_{lt} = (y_{\text{conv}} - m) \times n \tag{12}$$

where $m = (\mu/\alpha)$ and $n = (\alpha/\sigma)$. These two parameters can be precomputed offline.

D. Quantization Scheme

Previous works [10], [22]–[24] used the most common values of 16 or 12 b for both storage and computation precision of feature maps (activations). However, it is too luxurious to keep such a high precision for deep CNN models [12]. For the quantization of activations, the equal-distance nonuniform quantization (ENQ) scheme in [12] can reduce both the data width of computing components and the required DRAM bandwidth. In this paper, the ENQ scheme is employed.

TABLE I
QUANTIZATION RESULTS FOR BCNNS

Dataset/BCNN model	Original Accuracy	Quantization Accuracy[a]	Quantization Loss[a]
CIFAR-10/VGG-16	86.22%	84.87%	1.35%
CIFAR-100/VGG-16	56.34%	54.72%	1.62%
MNIST/LeNet-5	99.14%	98.37%	0.77%

[a] The results are gained with direct quantization, and accuracy loss is possible to be reduced by fine-tuning [26] [27].
[b] The BCNN models for CIFAR-10/100 and MNIST datasets are trained by the authors from scratch.

Note that only the outputs of ReLU layers are quantized and saved to DRAM. Hence, all of the quantized activations are greater than or equal to zero. For the ith layer of a CNN model, let us define two quantization parameters, i.e., $q_{m,i}$ and E_i, where $q_{m,i} > E_i$. With the ENQ scheme, an activation x in the ith layer is represented with the following equation:

$$\hat{x} = \begin{cases} \lfloor \dfrac{x}{2^{q_{m,i}-E_i}} \rfloor, & \text{if } x \le 2^{q_{m,i}} - 1 \\ 2^{E_i} - 1, & \text{otherwise.} \end{cases} \tag{13}$$

Thus, only E_i bits are required to store the activations of the ith layer. The dynamic range of the bit representation is from 0 to $2^{q_{m,i}} - 1$, and the resolution is $2^{q_{m,i}-E_i}$. $\hat{x}$ can be considered as an index of the quantized value. When a quantized activation needs to participate in the convolution computations after fetched from DRAM, it should be converted to a $q_{m,i}$ bits activation first. The optimal value of $q_{m,i}$ and E_i for each layer can be obtained through extensive simulations.

The proposed architecture sets $E_i = 6$ b uniformly to store the activations for all layers. To validate this quantization scheme for BCNNs, we perform experiments on several data sets, and the results are shown in Table I. One can see that the accuracy loss due to quantization is less than 1.7% even if the models have not been fine-tuned. The reasons why it works well are mainly of two folds. On one hand, binary weight training can be considered as both nonlinear projections and a regularizer [16], [25], which are able to improve the generalization ability of the model. Thus, the trained BCNN model is more robust to noise brought by quantization. On the other hand, the classification accuracy mainly depends on the relative magnitude difference between each output. The ENQ scheme can maintain most of the relative magnitude relationships between different activations. As a result, quantized BCNNs can still maintain the accuracy.

All computations in the BCNN's convolution phase are only additions and subtractions, so it is possible to directly use those indices with low data width (6 b + 1 sign bit) in convolution. Data are converted to its original precision for multiplication operations only when it comes across linear transformation operations. Therefore in the data path of convolution (i.e., compressor tree described in Section IV-C), we can

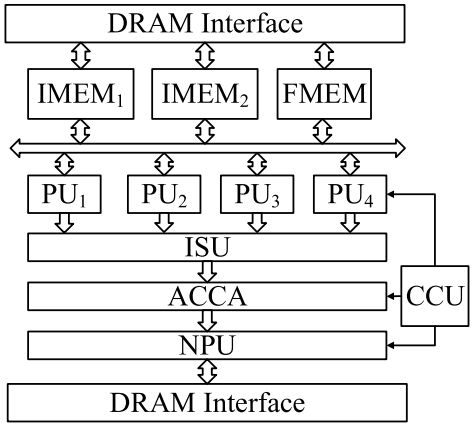


Fig. 2. Top-level block diagram of the proposed architecture.

TABLE II
SYMBOLIC REPRESENTATION FOR ARCHITECTURE ILLUSTRATION

Symbol	Description
C_{in}	# of input channels/feature maps
C_{out}	# of output channels/feature maps
p_{in} [a]	# of PUs, each PU process an input channel
p_{pu} [a]	# of activations an MFIR in a PU can process at a time
p_{out} [a]	# of output channels being calculated simultaneously at a time
w_{in}	width of input feature map
h_{in}	height of input feature map
w_{kernel}	width of binary kernel
h_{kernel}	height of binary kernel

[a] In this work, we set $p_{in} = 4$, $p_{pu} = 4$, $p_{out} = 32$.

employ computation components with less complexity rather than uniformly using high-precision adders or compressors.

IV. EFFICIENT HARDWARE ARCHITECTURE FOR BCNNS

A. Top-Level Architecture

In this section, an energy-efficient high-performance hardware architecture for BCNNs is proposed as illustrated in Fig. 2. For clear description, a symbolic representation is shown in Table II. Units in the diagram are described in the following.

1) Image memory (IMEM) is an on-chip dual-port SRAM. Each of these two IMEMs serves as a cache to save part of input feature maps, which is of size $C_{\text{in}} \times 2 \times w_{\text{in}}$. Here the factor 2 denotes the number of rows of each input feature map that is stored in one of the IMEMs.
2) Filter memory (FMEM) caches *all* binary parameters required by one convolutional layer. Note that the capacity of FMEM should be large enough to fit the largest convolutional layer.
3) Processing unit (PU) is the primary unit performing convolution operations. Each of the p_{in} PUs processes one input feature map, and the results should be combined. Detailed microarchitecture of PU is presented in Section IV-B.
4) The input feature map summation unit (ISU) performs the summation of all convolved input feature maps. Each of the p_{in} PUs processes an individual input feature map. To obtain an output neuron, the corresponding partial sums from different convolved input feature maps will be summed up in the ISU.
5) For a typical deep BCNN, there could be many input feature maps, which is more than the number of PUs. Hence an accumulation array (ACCA) followed by ISU will perform the accumulation of the partial sums in a partial parallel way.
6) The neuron PU (NPU) in Fig. 2 mainly deals with operations following the convolution process, including scaling (only for BWN), batch normalization, ReLU, and max pooling, which are mainly neuronwise data processing operations. It will generate $p_{\text{out}} \times 2$(output row) $\times$ $p_{\text{pu}} = 256$ output neuron concurrently per clock cycle.
7) The central control unit is the controller that schedules the processing flow of this architecture.

B. Processing Unit

Different layers in the same network can have different kernel sizes, such as 3×3 or 5×5. In spite of this, the state-of-the-art VGG-Net [4] shows that an homogeneous deep architecture with only 3×3 convolution kernels can achieve very high accuracy. Whether to support multiple kernel sizes mainly depends on the application. As mentioned before, the target application of the proposed design is low-power embedded systems, architectures that support only one kernel size cost less hardware resources and can be more energy efficient. Thus, in this paper, we mainly demonstrate the architecture supporting 3×3 kernels. The proposed architecture can be easily scaled to larger kernels with some modifications, which is described in Section IV-B2. For most of the CNNs, the step stride is usually set to 1. Other stride size, such as 2, can be achieved by adding a subsampling layer after the convolutional stage. In this design, we fix the stride to 1 and the PUs do not support different strides directly.

1) PU for 3×3 Kernels: The microarchitecture of the PU for 3×3 kernels is shown in Fig. 3. There are four data buffers (DBFs) for four input rows, respectively. Each DBF contains two columns of registers, where each column can save p_{pu} input activations. We simply refer them as left registers (LR) and right registers (RR) as shown in Fig. 3. Data can be loaded from IMEM_1 or IMEM_2 to RR through LR via on-chip data bus. The designs of LRs and RRs are mainly for the overlapping of convolution operation and data prefetching, which is shown in Section IV-G.

The PU convolves input data in RRs with binary kernels of all output channels. For each row of an input feature map, convolution operation can be described as

$$y(n) = \sum_{i=-[\frac{w_{\text{kernel}}}{2}]}^{[\frac{w_{\text{kernel}}}{2}]} x(n+i)k(i) \tag{14}$$

where x refers to the input data and k is a binary weight. Multiple-input multiple-output finite impulse response (MFIRs) filters perform convolution operations for p_{pu} input data in RRs. The number of stacked MFIRs indicates the parallelism of this architecture: multiple MFIRs convolve

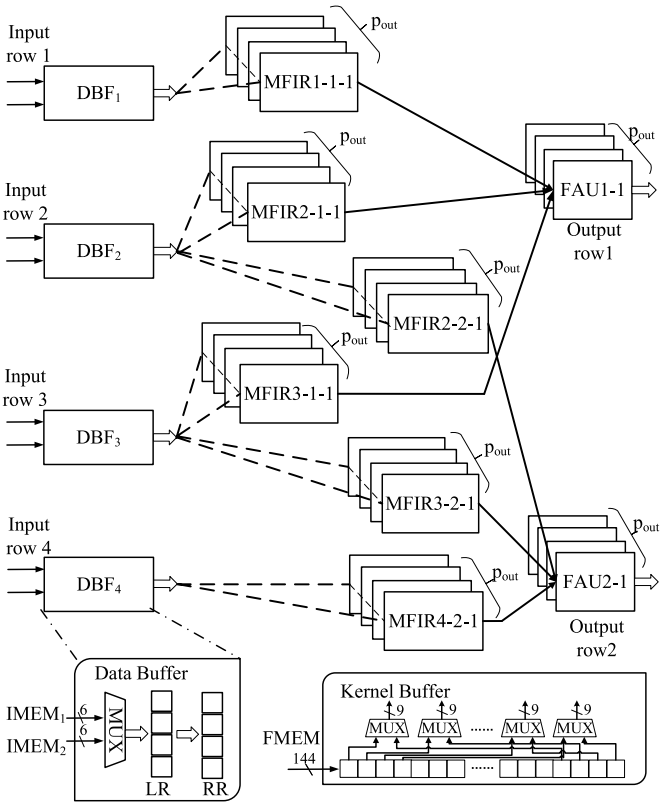


Fig. 3. Detailed PU. The MFIRm-n-k unit denotes an MFIR unit that uses the input from the mth row to calculate the nth output row, and $k = 1, 2, \ldots, p_{\text{out}}$ is the index of MFIRs and output channels. The FAUn-k combines the outputs of three MFIRs to calculate the nth output row, and $k = 1, 2, \ldots, p_{\text{out}}$ is the index of FAUs and output channels.

the same input activations with weights of different output channels, so that it can reuse the input data. We describe this parallelism as p_{out}, which denotes the number of output channels that are calculated together. In this design, $p_{\text{out}} = 32$ and $p_{\text{pu}} = 4$.

For 3×3 kernels, outputs of three MFIRs for the same output channel shall then be summed together in fast adder units (FAUs) as Fig. 3 shows. In the second and third rows, there are $2 \times p_{\text{out}}$ MFIRs, which is different from the first and fourth rows. This is because for 3×3 kernels, the second and third input rows are shared by two output rows while using different parts of the same kernels. Rather than directly performing summations in MFIRs and FAUs using adder trees, we develop an optimized compressor tree structure based on dedicated 4:2 and 3:2 compressor circuits, which is discussed in detail in Section IV-C.

The PU also contains a kernel buffer composed of registers, which can buffer C_{out} kernels from FMEM and send kernels to corresponding MFIRs through multiple multiplexers.

2) Support for Different Kernel Sizes: The microarchitecture of a PU for 3×3 kernels can be modified to support larger kernel sizes. To support $k \times k$ kernels, the modifications are as follows.

1) The number of DBFs in PU should be increased to $k+1$.
2) Each of the added DBFs should also be connected to another $2 \times p_{\text{out}}$ MFIRs as in 3×3 kernels.

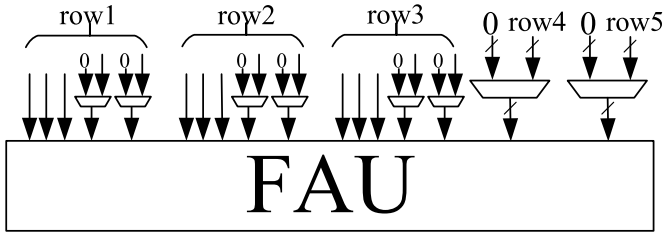


Fig. 4. Example to show how to scale the FAU to support 3×3 and 5×5 kernel simultaneously.

3) The compressor trees (FAUs) should be modified to accumulate $k \times k \times p_{\text{pu}}$ input data.

With these modification, the PU can process more input rows to support larger kernel size while still generating two output rows.

One example of the FAU to support both 5×5 and 3×3 kernels is shown in Fig. 4. If the kernel in one layer is 3×3, all input activations from the fourth and the fifth rows should be taken place of by zeros, and the fourth and the fifth input activations in the first three rows should also become zero. In this way, this architecture is able to support different kernel sizes through hardware reuse.

C. Compressor Tree

The main operations of CNNs are MACs. BCNNs have already removed all of the multiplications in convolution process, so the critical path lies in the accumulation part. In this design, there are four PUs and the designed kernel size is 3×3. Hence in one convolution phase of obtaining an individual output neuron, $w_{\text{kernel}} \times h_{\text{kernel}} \times p_{\text{in}} = 36$ data are to be added together. However, such a long path results in a long delay and limits the frequency of the system if it is implemented with an adder tree. In this paper, an optimized compressor tree based on dedicated 3:2 and 4:2 compressors is used to improve the system performance. The block diagram is shown in Fig. 5.

A compressor [28] is composed of several 1-b full adders as illustrated in Fig. 6(a) and (b). For a 3:2 compressor, it satisfies

$$X + Y + Z = S + C \times 2 \tag{15}$$

where X, Y, and Z are the three inputs of the compressor, and S and C are the compressor's outputs. All of them have N bits as shown in Fig. 6(a).

The structure of a 4:2 compressor is similar to 3:2 compressor as shown in Fig. 6(b). For simplicity, only an 1-b 4:2 compressor is shown. With C_{out} of the $(k-1)$th bit being connected to C_{in} of the kth bit, multiple 1-b 4:2 compressors constitute a multibit compressor. The input–output relations of a multibit 4:2 compressor is

$$X + Y + Z + W + \text{C}_{\text{in\_0}} = \text{Carry} \times 2 + \text{Sum}. \tag{16}$$

The delay of a 4:2 compressor is twice of that of a 1-b full adder. There is no carry chain in 3:2 and 4:2 compressors. Thus, it is a great deal of reduction in data path delay

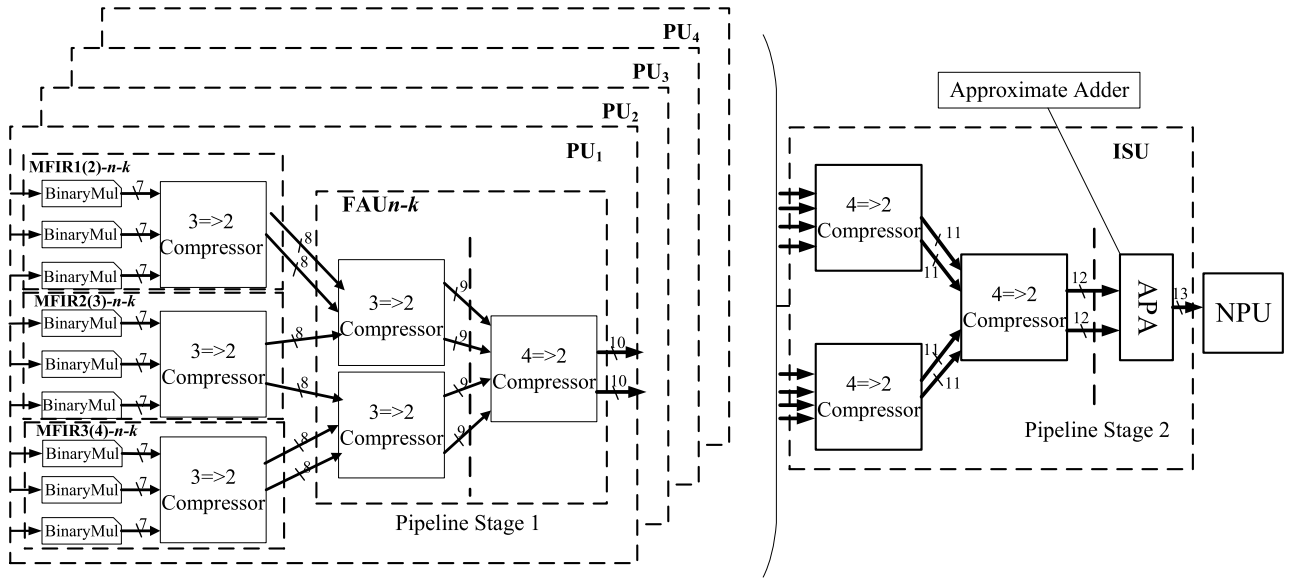


Fig. 5. Compressor tree for an output neuron of one output channel. It stretches across part of PU and ISU. The indices n and k both have the same meaning as in Fig. 3.

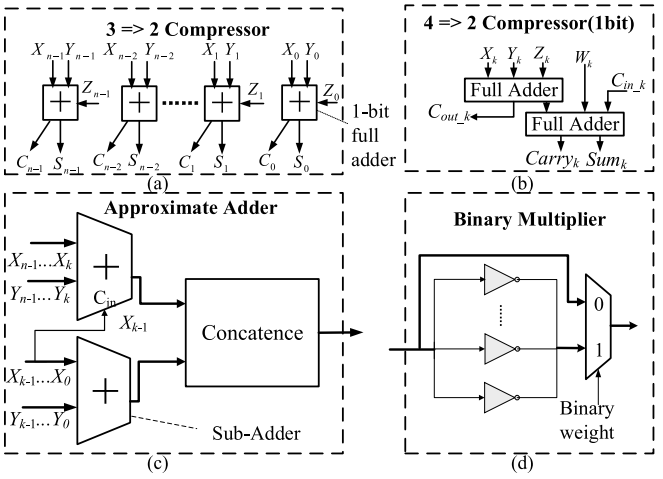


Fig. 6. Details of each unit in a compressor tree. (a) 3:2 compressor. (b) 1-b 4:2 compressor. (c) Approximate adder in ISU and ACCA. (d) Optimized binary multiplier in MFIRs.

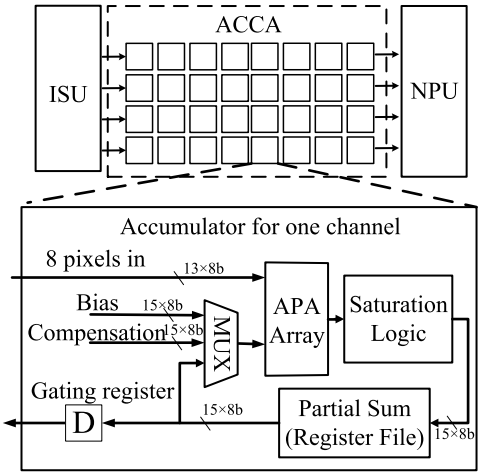


Fig. 7. Diagram of Accumulate Array.

compared with a multi-input adder which has long carry propagation for several stages.

Instead of using a traditional adder tree, we propose to cascade multiple 3:2 and 4:2 compressors together in a tree structure. It is able to compress the summation of 36 data into 2 data. At the end of the the compressor tree, an approximate adder will add two data up, as shown in Fig. 5. The approximate adder will be discussed in detail in Section IV-E. The compressor tree can significantly cut down the critical path.

D. Approximate Binary Multiplier

As described in Section III-A, the multipliers are replaced by the optimized approximate binary multiplier shown in Fig. 6(d). Synthesis results show that, compared with the binary multiplier implementation using two's complement in [19], the area of the optimized one is reduced by 60% by removing the add-one adder. Consider that there are $2p_{\text{pu}}p_{\text{in}}p_{\text{out}}w_{\text{kernel}}h_{\text{kernel}} = 9216$ binary multipliers in the design, hence 9216 adders are saved. The compensation is achieved in the ACCA as shown in Fig. 7.

E. Approximate Adder

As shown in Figs. 5 and 7, for each output neuron, there are only two adders left in the accumulation path, which have the largest data width as well as the longest data path delay. Deep neural networks trained with weight projections or quantization (especially binarization) exhibit a remarkable robustness to various kinds of distortions (noises) [25]. Inspired by this, we developed a dedicated approximate adder to replace those adders in the data path, which is shown in Fig. 6(c).

A significant portion of the power consumption of an adder is due to the glitches that are caused by carry propagation.

TABLE III
COMPARISON WITH OTHER 15-B APPROXIMATE ADDERS

	area (μm^2)	delay (ps)	power (μW)	Energy (pJ)	PDAP ($pJ \cdot \mu m^2$)	Loss on CIFAR-10	Loss on CIFAR-100
ETAI	298	436	79.3	0.0350	10.43	4.64%	5.37%
KimA	430	491	123	0.0610	26.23	0.17%	0.44%
LOA	247	428	68.5	0.0294	7.262	74.87%[a]	53.72%[a]
LUA	424	490	123	0.0603	25.56	74.87%[a]	53.72%[a]
Proposed	307	401	86.1	0.0344	10.56	-0.61%[b]	0.30%

[a] The classification accuracies of LOA and LUA on two datasets decrease to only 10% and 1%, respectively.
[b] With the proposed adder, the classification accuracy even increase on the CIFAR-10 dataset.

To relieve the effect of carry propagation, the proposed approximate adder is designed as follows: an N-bit adder is split into two subadders at bit k. For the higher $(N-k)$-bits subadder, its input carry bit C_{in} is approximately speculated using the kth bit of one of the input data. This structure can reduce not only data path delay but also hardware complexity, because the carry of the approximate adder ripples through a shortened path.

When the k is set to half of the word size, the hardware efficiency gain can reach the maximum. However, error rate (accuracy loss) increases as the value of k becomes larger. The optimal value is chosen at the inflection point of BCNN's accuracy loss. Through experiments on several data sets in the design time, it is able to find the optimal one to balance the hardware efficiency gain and the accuracy loss among all k values.

For the purpose of comparison, we implemented four previously discussed approximate adders, which are lower-part-or adder (LOA) [29], Error-Tolerant Adder: I (ETAI) [30], Lu's Adders (LUA) [31], and KimA [32], to explore the efficiency of different approximate adders in the proposed BCNN architecture. All of them were synthesized using TSMC 90-nm technology, and the timing/area constraints were all set to the same. Besides, most of the existing works share the same idea that splitting the adder into different parts can reduce the carry propagation chain. Thus, we set the splitting position of these adders to the same for fair comparisons.

The comparison results are shown in Table III. For brevity, only the hardware cost of $k = 4$ for all adders are reported. We define the power-delay-area product as PDAP to evaluate the power/area gain together. The smaller the PDAP is, the more power/area saving is gained. Some of the approximate adders like LOA and LUA cannot work on the proposed BCNN architecture, since their unipolar errors go to negative infinity when accumulated through layers. Finally, all activations become zero due to ReLU layers. Other adders have various error reduction mechanisms, which can improve the accuracy to some extent. However, these mechanisms increase the hardware complexity.

As for the proposed approximate adder, it is easy to find that the possible errors are $\pm 2^k$, where k is the split position. Both of the probabilities of incurring these two errors are $1/8$ if we assume the kth bit of two inputs and the carry from the $(k-1)$th bit conform to uniform distribution. Thus, the expectation of the error caused by the proposed adder is zero. We have the prior knowledge that the add operations

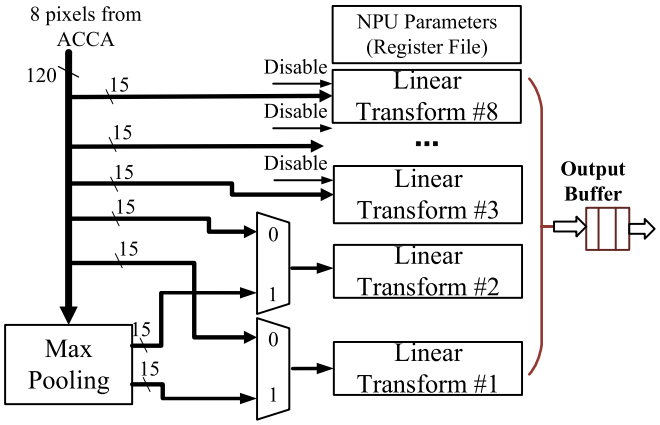


Fig. 8. Detailed architecture of a slice of the NPU.

using the approximate adders in the proposed BCNN architecture are used for data accumulation. According to the central limit theorem for independent and identically distributed random variables, the total accumulated error will approximately conform to zero mean Gaussian distribution. Hence, the error of most of the activations will be limited in a small range. In other words, most of the positive and negative errors of the proposed adder can be canceled out. Besides, the simple carry speculation mechanism costs no extra hardware resource for error reduction. As a result, our proposed adder is comparable to ETAI and LOA with respect to area, power, and delay, while having low classification accuracy loss.

F. NPU and Linear Transformation Unit

The NPU of the proposed architecture applies four main operations to each convolved data from ACCA, namely, scaling, batch normalization, ReLU activation, and max pooling. The detailed architecture of a slice of NPU is illustrated in Fig. 8. Eight convolved neurons, that is, two rows of an output channel with four neurons per row, flow into one slice of the NPU, and each of them passes through linear transformation operations which are mentioned above. The order of pooling and linear transformation is exchanged according to Section III-C. Adding an additional "Disable" signal to block the data transformation in part of the linear transformation units can reduce the dynamic power dissipation.

For linear transformation, computation order is changed according to Section III-C. Besides, we also propose the

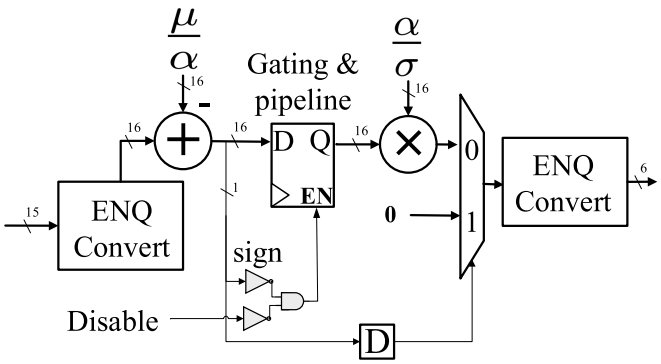


Fig. 9. Linear transformation unit.

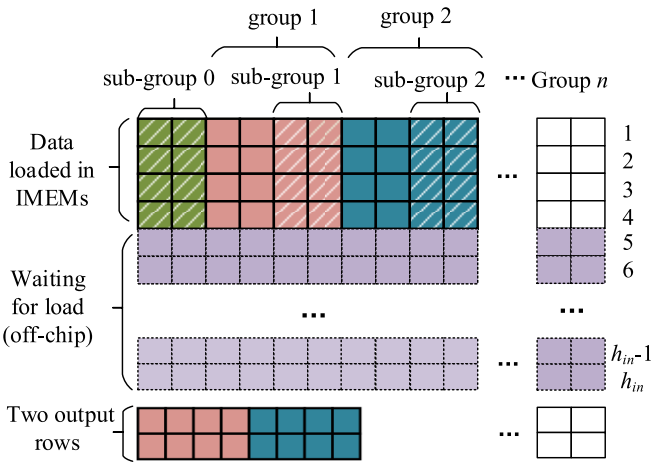


Fig. 10. Data processing sequence of proposed architecture.

negative skipping scheme as follows. Due to the existence of the ReLU activation function, statistically half of the output of certain stages will be zero. To further reduce the dynamic power and cut down the path delay of linear transformation units, a gating register is inserted in front of the multiplier as shown in Fig. 9. If the output of the adder is negative, its sign bit will block the data from being further processed, and the linear transformation unit will directly output zero. Nearly half of the dynamic power dissipation in the NPU can be saved because there are less data switching in the multiplier, which consumes most of the energy in this unit. Besides, the gating register also reduces the data path delay and can block the unnecessary data abandoned by earlier pooling.

G. Processing Schedule for Convolution

1) Dataflow: For the proposed schedule, the convolution of a feature map is divided into consecutive convolution rounds (CRs), where four rows of p_{in} input feature maps are processed in parallel during each CR. The convolution is carried out in a group-by-group way, where each group requires $rows \times p_{\text{pu}} \times p_{\text{in}} = 4 \times 4 \times 4$ input activations. Fig. 10 illustrates the data processing schedule of convolution on the proposed architecture. For simplicity, only one input channel and one output channel are shown. The squares with colors stand for activations in the feature map, and different colors mean they are processed or outputted in different iterations of a CR. Some of those activations should be buffered during computation and they are drawn with slashes.

Taking the convolution of a feature map f in one CR as an example, in the initialization step, for each row being processed, p_{pu} activations and another two preloaded activations, which are drawn as group 1 and subgroup 0, respectively, will be convolved with binary kernel of the first output channel, and eight activations (two rows with p_{pu} activations per row) will be outputted. Then the same operations will be carried out for all output channels, so that the partially convolved maps of the first $2p_{\text{pu}}$ output activations related to group 1 and subgroup 0 of feature map f are calculated. Because in each PU, there are p_{out} MFIRs which convolve the same input row concurrently, this process lasts for $\lceil (C_{\text{out}}/p_{\text{out}}) \rceil$ clock cycles. After that, the convolved input activations of the first two rows are of no use. After finishing the processing of group 1 and subgroup 0, the last two processed input activations of each row, that is, subgroup 1, will be buffered for reuse in the process of the following activation group, where activations from group 2 and subgroup 1 are convolved with corresponding kernels. This kind of operations will keep going until all input activations of four rows, that is, group 1 to group n, are processed and the current CR is finished.

For each output channel, there are C_{in} computed temporary convolved maps (TCMs). Each output feature map is the element-wise summation of corresponding C_{in} TCMs as shown in (4). Since there are p_{in} PUs in our architecture and each PU processes one input channel, p_{in} input feature maps can be convolved in parallel in one CR. During each CR, p_{in} partial TCMs (PTCMs) are outputted, where each PTCM has two rows of activations. With the parallelism of p_{in} PUs, when a CR is finished, p_{in} PTCMs for each output channel are produced and summed together in an element-wise way. Then the design will continue to produce next p_{in} PTCMs for each output channel, until C_{in} PTCMs for each output channel are produced. Accumulating those PTCMs for each output channel in an element-wise way, the final values of two rows of each output feature map are obtained. Using the same method to process all input activations in a four-row-by-four-row way, the convolution in a convolutional layer is completed.

During the convolution operation, each input activation is first loaded from DRAM to the on-chip SRAMs (IMEMs), and then loaded from IMEMs to the LR (see Section IV-B) in PU. After all LRs have been filled up, activations are shifted from LRs to RRs for convolution. With the proposed processing schedule, convolution and data loading can be interwoven in the following two aspects to reduce the instantaneous I/O bandwidth requirement.

1) For each activation in the first two rows in a CR, which is in the rth row in the input feature map, after it has been fetched to the PU, it will never be fetched from SRAM again. Thus, the activation in the same location of the $(r+4)$th row, which will be convolved in the next four-row processing, can be preloaded from DRAM to IMEM. The DRAM preloading operations can overlap the convolution course.

2) After the tth, $(t+1)$th, …, $(t+p_{\text{pu}}-1)$th activations in four input rows have been shifted from LRs to RRs, $\lceil (C_{\text{out}}/p_{\text{out}}) \rceil$ cycles are needed before these activations are used up. Thus, during this time interval, LRs become available and can be prefilled up using the $(t+p_{\text{pu}})$th,…,$(t+2p_{\text{pu}}-1)$th activations in the same four rows.

2) Data-Reuse and DRAM Access Analysis: The goal of the presented energy-efficient BCNN processing schedule is to perform most data accesses using the data paths with low energy cost. To solve the limitation of on-chip memory capacity, three levels of storage hierarchy are used in the proposed architecture. Sorting their energy cost for data access from high to low, it includes DRAM, SRAM, and registers.

In the processing schedule elaborated in Section IV-G, we make efforts to maximize the input data reuse in the lowest energy cost registers and minimize the number of DRAM accesses.

In the basic stage of BCNNs, each input activation will be loaded from off-chip DRAM to on-chip SRAM (IMEMs) for only one time, and loaded from on-chip SRAM to registers (LRs) for no more than two times. The number of times that one input activation is fetched from LRs to RRs equals 2. Therefore, each input activation is mostly read from RRs, which cost least energy, by MFIRs for C_{out} times. The number of times that an activation is written to DRAM equals the number of times it is read from DRAM.

The DRAM dominates the overall memory accessing power, which is about 33 times more than that of SRAM [8]. To show the benefit brought by performing the novel processing schedule proposed in Section IV-G, here we make a rough comparison of DRAM accessing times with the processing schedule of YodaNN [19]. Running a convolutional layer with C_{out} output channels on YodaNN, each time an individual input activation x will be fetched to on-chip IMEM (SRAM) from DRAM to obtain the outputs of n_{ch} output channels,[1] and then x will be overwritten by other activations of the same input feature map. In order to calculate for entire C_{out} output channels, this process will repeat for $\lceil (C_{\text{out}}/n_{ch}) \rceil$ times, and it is also the number of times accessing DRAM for x. In contrast, with the processing schedule proposed in this paper, an input activation is fetched from DRAM for only one time.

As for the weights of BCNNs, the proposed architecture chooses to cache all weights and other parameters of one stage on-chip before it starts the computation. Thus, during one inference time, each weight is loaded from DRAM to SRAM only once.

V. Implementation Results and Comparisons

A. Evaluation Metrics

1) Computation Complexity: Additions and binary multiplications are considered as two separate operations according to [19]. Therefore, the number of operations for a basic stage of a BCNN can be calculated as follows:

$$2C_{\text{in}}C_{\text{out}}h_{\text{kernel}}w_{\text{kernel}}h_{\text{out}}w_{\text{out}}+2h_{\text{out}}w_{\text{out}}C_{\text{out}}(\text{Ops}) \quad (17)$$

[1]In YodaNN, IMEM is used to cache the inputs. The n_{ch} is the design parameter of YodaNN, and can be set to 8, 16, or 32.

where $h_{\text{out}} = h_{\text{in}} - h_{\text{kernel}} + 2P + 1$ and $w_{\text{out}} = w_{\text{in}} - w_{\text{kernel}} + 2P + 1$ denote the height and width of an output feature maps with zero padding P on the border, respectively. The first term of (17) is the computation complexity of convolution, and the second one relates to the scaling and batch normalization.

2) Performance/Throughput: The peak performance of our architecture is given by

$$\Theta_{\text{peak}} = 2h_{\text{kernel}}w_{\text{kernel}}p_{\text{in}}p_{\text{out}}p_{\text{pu}}r_{\text{out}}f \quad (18)$$

where r_{out} is the number of rows being outputted at a time and f is the clock frequency.

We define the effective performance of our architecture in this way: run the proposed architecture with a deep BCNN benchmark and record the number of clock cycles to finish a complete inference. The effective throughput is

$$\Theta_{\text{effective}} = \frac{\Sigma_{\text{stage}}\zeta}{\Sigma_{\text{stage}}T} \quad (19)$$

where ζ is the computation complexity of a certain stage and T is the time consumed in the corresponding stage calculated using the required clock cycles and the clock frequency.

3) I/O Bandwidth and I/O Power Dissipation: As discussed in Section IV-G, I/O requirements consist of two parts during one inference time. One is exchanging feature map data during computation, and the other is loading parameters of the certain stage for only one time.

During the computation of a basic stage of a BCNN, there are $N_{\text{IO\_fm}}$ feature map data and $N_{\text{IO\_weight}}$ kernels passing through the DRAM bus, where

$$N_{\text{IO}\_fm} = w_{\text{in}}h_{\text{in}}C_{\text{in}}(\text{read})+w_{\text{out}}h_{\text{out}}C_{\text{out}}(\text{write}) \quad (20)$$

$$N_{\text{IO\_weight}} = C_{\text{in}}C_{\text{out}}. \quad (21)$$

So the required I/O bandwidth

$$\text{BW}_{\text{IO}} = \frac{N_{\text{IO}\_fm} \times n_{fm} + N_{\text{IO\_weight}} \times n_{\text{kernel}}}{T_{\text{stage}}}(\text{bits/s}) \quad (22)$$

where n_{fm} and n_{kernel} denote the number of bits used to store a feature map data and a kernel, respectively.

It is important to stress the fact that the power of access to external memory should not be neglected and that the I/O power dissipation related to this design is proportional to the required I/O bandwidth. Let P_{core} and P_{IO} be the core power and the power of accessing DRAM, respectively.

Thus, the energy efficiency is

$$H_E = \frac{\Theta_{\text{effective}}}{P_{\text{core}} + P_{\text{IO}}}. \quad (23)$$

B. On-Chip Memory Capacity and Precision of Computing Components

In this design, the input feature map data stored in IMEMs uses only 6 b. Thus, the data widths of compressors in the compressor trees start from 6 b and gradually increase to prevent overflow. The data width of the approximate adders in the ISU is 13 b, and the split position k for them is 3. The approximate adders in the ACCA have their width up to

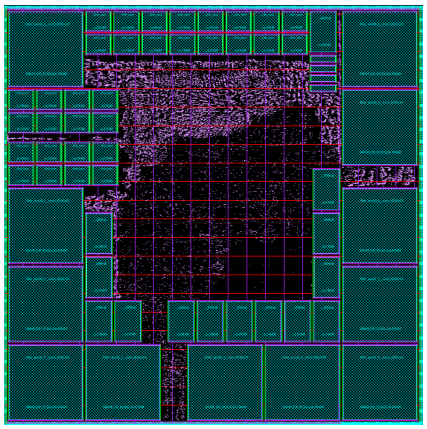


Fig. 11. Layout of the proposed design under SMIC 130-nm process.

15 b with the corresponding split position k set to 4. These two values of k are selected through extensive experiments as described in Section IV-E. Besides, the computational precision in the NPU is 16 b. The NPU parameters (μ/α) and (α/σ) are both quantized to 16 b.

The capacity of each IMEM is 21.6 KB so that two IMEMs can store C_{in} partial input feature maps together, where each partial feature map has four rows of the original feature map. This capacity is sufficient to store 57344 neurons. For larger input images (feature maps), data should be partitioned in advance. FMEM is 295 KB, which is sufficient for most of the state-of-the-art deep models to cache all binary weights of an arbitrary convolutional layer. The scratch pad in each slice of ACCA has a capacity of 1.68 KB, and there are 32 slices. Besides, there is also a register file of 2 KB in NPU to store the related parameters.

C. Implementation Summary and Comparisons

The design is coded with register transfer level and verified against the software model of the proposed architecture. Then we synthesized it under the SMIC 130-nm standard cell library using the Synopsys Design Compiler with worst case process-voltage-temperature (PVT) corner and placed and routed the design using the Synopsys IC Compiler. The layout can be seen in Fig. 11. Table IV lists the implementation results of the proposed architecture, with design parameters p_{out}, p_{in}, p_{pu}, h_{kernel}, and w_{kernel} set to 32, 4, 4, 3, and 3, respectively. The total area is 44.9 mm^2, where the FMEM occupies about half of the total area. The core power is 694.8 mW at a clock frequency of 190 MHz. Table IV also shows other metrics, such as effective performance and I/O power, which is based on the benchmark test that will be described in detail later. According to (18), the peak performance is up to 3.50TOp/s. Besides, the same design has also been synthesized (not placed and routed) under the Taiwan Semiconductor Manufacturing Company 90-nm technology with typical PVT corner, which has the maximum frequency of 650 MHz. This frequency gap indicates that the performance of the proposed architecture can be further improved when implemented with more advanced technology and better layout design.

We benchmark the proposed architecture using the binarized version of a VGG-16 model on the ImageNet data set, and the timing and bandwidth results are shown in Table V. Due to limited computational resources, we have not trained a binary weight VGG-16 model on ImageNet with high accuracy. However, the throughput and I/O bandwidth are independent of model accuracy. Therefore, we apply a VGG-16 model with random binary weights and images from ImageNet data set to evaluate the performance, and the results are still trustworthy. The accuracy of BCNNs on ImageNet can be checked in other works [18], [33]. The inputs of the first stage are $224 \times 224 \times 3$ RGB images. The table contains only basic stages, which are the most computational intensive. The fully connected layers are not considered. Only 35.10 ms are required to complete an entire convolution inference. Thus, it can achieve 28.5 fps. The average throughput is about 875.6GOp/s, and it is about two times better than that of the YodaNN [19] even if the working frequency is no more than half of that of YodaNN. On CIFAR-10 data set (inputs are $32 \times 32 \times 3$ RGB images) with the same BCNN model, our design can achieve 442.68 ft/s.

In Table V, one can find that the throughputs differ from layer to layer, which means in some layers the accelerator suffers from core underutilization. Actually, for the proposed design, underutilization occurs when C_{in} and C_{out} do not match with the design parameters of p_{in} and p_{out}. For example, the throughput becomes lower when running the first stage because one of the PUs is idle. The proposed design can enjoy high resource utilization rate when the number of input–output channels are multiples of p_{in} and p_{out}. In the state-of-the-art deep CNN models like AlexNet, VGGNet, and ResNet, the number of feature maps are commonly set to the power of 2 and mostly greater than 32; hence, we consider $p_{\text{out}} = 32$ as an optimal design parameter to enjoy high core utilization rate. Taking the fact that the first stage of CNNs usually has three channels into consideration, p_{in} should not be very large. Besides, the value of p_{in} also affects the area of the core significantly. Thus, we set $p_{\text{in}} = 4$ to balance the throughput, core utilization rate, and core area.

TABLE IV
IMPLEMENTATION RESULTS OF THE PROPOSED DESIGN

Characteristics	Results
Technology	130nm
Type	Layout
Core Voltage	1.08V
Area	$6.71 \times 6.70\text{mm}^2$
On-chip Mem	393KB
Clock Frequency	190MHz
Average I/O Bandwidth	0.44GB/s
Power(core)[a]	694.8mW
Power(I/O)[b]	73.92mW
Performance/Throughput	**Results**
Peak Performance	3.50TOp/s
Effective Performance	875.6GOp/s
Energy Efficiency	1.14TOp/s/W
Benchmark Model	VGG-16
Frame Rate Per Second	28.5 for $224 \times 224 \times 3$ image

[a] The core power is estimated based on the post-layout simulation.
[b] The I/O power is estimated at a bandwidth of 0.44GB/s using LPDDR3 memory.

TABLE V
THROUGHPUT AND EFFICIENCY FOR INDIVIDUAL STAGES OF BENCHMARK VGG-16 MODEL AT 190 MHZ

Stage index	$(C_{in}, C_{out}, w_{in})$	# Operations	# Clock	time	Throughput	I/O BW requirement
1	(3, 64, 224)	183.04MOp	62160	0.34ms	549.67GOp/s	7.37GB/s
2	(64, 64, 224)&pool	3709.01MOp	798000	4.17ms	882.33GOp/s	0.72GB/s
3	(64, 128, 112)	1854.50MOp	373912	1.94ms	941.53GOp/s	0.93GB/s
4	(128, 128, 112)&pool	3704.19MOp	741272	3.86ms	949.03GOp/s	0.39GB/s
5	(128, 256, 56)	1852.10MOp	375116	1.94ms	937.70GOp/s	0.48GB/s
6	(256, 256, 56)	3701.78MOp	746956	3.89ms	941.40GOp/s	0.32GB/s
7	(256, 256, 56)&pool	3701.78MOp	746956	3.89ms	941.40GOp/s	0.21GB/s
8	(256, 512, 28)	1850.89MOp	424550	2.22ms	828.16GOp/s	0.27GB/s
9	(512, 512, 28)	3700.58MOp	847462	4.45ms	829.58GOp/s	0.20GB/s
10	(512, 512, 28)&pool	3700.58MOp	847462	4.45ms	829.58GOp/s	0.15GB/s
11	(512, 512, 14)	925.15MOp	247723	1.30ms	709.50GOp/s	0.34GB/s
12	(512, 512, 14)	925.15MOp	247723	1.30ms	709.50GOp/s	0.34GB/s
13	(512, 512, 14)&pool	925.15MOp	247723	1.30ms	709.50GOp/s	0.30GB/s
Total/Average	-	30.74GOp	6671625	35.10ms	875.6GOp/s (Average)	0.44GB/s (Average)

[a] Each stage includes convolutional, ReLU and batch normalization layers. An max pooling layer only exists in certain stages.

TABLE VI
COMPARISON WITH PREVIOUS WORKS THROUGH TECHNOLOGY AND FREQUENCY SCALING

Metrics	Benchmark CNN Model	CNN Model Type	Freq (MHz)	Core Voltage (V)	Peak Perform (GOp/s)	Effective Perform (GOp/s)	Core Power (mW)	I/O Power (mW)	Total Power (mW)	Energy Efficiency (GOp/s/W)	Area (mm^2)	Area Efficiency (GOp/s/mm^2)
FINN-CNV(FPGA) [20]	VGG-16	Binary	200	-	-	2465	3600	-	11700	210	-	-
Alemdar's (FPGA) [34]	LeNet-5	Ternary	200	-	-	864	2760	-	-	-	-	-
DLAC(14nm) [35]	ResNet-34	Ternary	500	-	2500	1340	-	-	-	-	1.09	600.0
YodaNN(65nm) [19]	ConvNet[c]	Binary	400	1.2	1510	423	153	328[a]	481	879[e]	3.11	136.0[e]
this work(130nm)	VGG-16	Binary	190	1.08	3501	876	694.8	73.9[a]	768.7	1139	44.92	19.5
this work(65nm)[b]	VGG-16	Binary	380	1.08	7002	1752	694.8	147.8	842.6	2079	11.23	156.0
Comparison[b]	-	-	-	-	4.63x	4.14x	4.54x	0.45x	1.75x	2.37x	3.61x	1.15x

[a] We estimate the I/O power of our work in the same way with YodaNN: simply estimate it with 28nm LPDDR3 memory consuming 21 pJ/bit according to [10].
[b] We only compare our work with YodaNN in 65nm because other works are different in terms of either CNN model type (Ternary vs Binary) or implementation method (FPGA vs ASIC).
[c] The benchmark CNN model used in YodaNN is from [10]. It is a 3-layer model with 7×7 kernels. The number of input/output channels are set as (3,16), (16, 64), (64, 256), respectively.
[d] The datasets used for test in all other architectures and this work are CIFAR-10, MNIST, ImageNet, Stanford backgrounds and ImageNet, respectively. We have not verified the classification accuracy of binary weight VGG-16 models on ImageNet due to limited computational resources. However, the throughput/power/area numbers in the table are independent of the classification accuracy, so they make sense in general.
[e] In the original paper, YodaNN's energy/area efficiency numbers are calculated based on its peak performance, and its I/O power is excluded. In this paper, we recalculate the energy/area efficiency of YodaNN based on its actual performance and take the I/O power into consideration.

The I/O power in Table IV is estimated with an average I/O bandwidth of 0.44 GB/s according to Table V. We use the same I/O power estimation method shown in [10] and [19], which assumes an implementation of 28-nm LPDDR3 memory and estimates the energy usage of 21 pJ/b. The I/O power of this design is only 73.92 mW with a clock speed of 190 MHz, while YodaNN [19] costs 328 mW at 400 MHz. According to (23), the proposed architecture achieves not only high throughput, but also spectacular energy efficiency of 1.14TOp/s/W. As shown in Table V, it is found that I/O bandwidth requirement differs from stage to stage. In general, it decreases as the size of the input–output feature maps becomes smaller at the beginning, but it then starts to increase because the requirement of transferring binary weights begins to dominate when the number of input–output channels is larger. It is worth mentioning that when going through the first stage of VGG-16 Net, the proposed architecture suffers from large I/O bandwidth requirement and relatively low throughput. The reasons are mainly of two folds.

1) The number of input channels of the first stage is less than the number of PUs, which makes one of the PUs idle and the input data of this PU should be forced to zero. It lowers the PU efficiency and throughput.
2) The number of output channels in the first stage is many times larger than that of the input feature maps, and the proposed architecture can finish data processing of the first stage within only 0.34 ms. This results in high data/time ratio and very high instantaneous I/O bandwidth requirement.

Although the instantaneous I/O bandwidth requirement of the first stage is high, it is still under the maximum DRAM bandwidth (12.8 GB/s for DDR3).

BCNNs have its weights constrained to only $+1$ or -1, resulting in higher throughput and energy efficiency than original CNN models according to the results shown in YodaNN [19]. So our work and those architectures for high-precision CNNs are not comparable. The comparisons of our work with the existing state-of-the-art accelerators for low-precision CNN models are presented in Table VI.

To estimate the core power, maximum frequency and area under the 65-nm CMOS technology for fair comparison, we assume that the width, length, and thickness of gate oxide

of MOS transistors are scaled by the same factor. Following equations are used for technology projection with fixed supply voltage

$$s = \frac{l_{\text{old}}}{l_{\text{new}}}, \quad P_{\text{new}} = \frac{P_{\text{old}}}{s}, \quad S_{\text{new}} = \frac{S_{\text{old}}}{s^2}$$
$$\tau_{\text{GD,new}} = \frac{\tau_{\text{GD,old}}}{s}, \quad f_{\text{max,new}} = f_{\text{max,old}} \times s \tag{24}$$

where s represents the scaling factor of two different technologies, l is the channel length, P denotes the power consumption and S corresponds to area. The gate delay, denoted as τ_{GD}, is scaled down by a factor of s, while the maximum clock frequency f_{max} is scaled up by the same factor.

As shown in Table VI, the proposed architecture has better throughput and energy efficiency compared with existing works. When working at 380 MHz in 65-nm technology, which is nearly the same as the working frequency of YodaNN, the I/O power of the proposed architecture is only 45% of that of YodaNN. Although the total power of the proposed architecture is 1.75 times of that of the YodaNN, the throughput of the proposed architecture is 4.14 times higher. Thus, the proposed architecture is more energy efficient. When scaled to 65-nm technology, the energy efficiency of the proposed architecture increases to 2079GOp/s/W, which significantly outperforms other low-precision CNN implementations. The area of the proposed design is larger than that of YodaNN, since more on-chip SRAMs are used. However, the area efficiency of our design is better. Besides, YodaNN only supports convolutional layer. Our design can support not only convolutional layers but also pooling and batch normalization layers. The implementation of these layers consumes more area. The implementation results show that the proposed architecture is applicable in low-power embedded systems.

It should be mentioned that such high energy efficiency can be attributed to the following factors.

1) Various architectural level and circuit level optimizations make the computations more efficient.
2) Error resilience of BCNNs is leveraged to reduce the hardware complexity.
3) The processing schedule exploits utmost data reuse and mostly accesses data from registers, which significantly reduce the power dissipation on chip.
4) The proposed architecture stores all weights of a convolutional layer on-chip to dramatically reduce the I/O cost for weight exchanging.
5) The I/O power dissipation and bandwidth requirements are significantly reduced due to the proposed I/O interwoven scheme together with memory efficient quantization scheme discussed above.

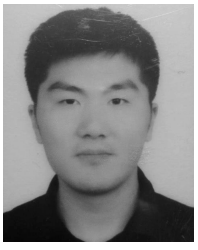
VI. Conclusion

In this paper, we have presented a high-performance, energy-efficient BCNN architecture by taking advantages of binary weights and other properties of BCNNs. The proposed architecture exploits utmost data reuse through well-optimized processing schedule, leading to drastically reduced I/O access and power. Also many innovations at microarchitectural level are incorporated, including pipelined compressor trees, binary multipliers with two compensation schemes, and dedicated approximate computing units. Besides, algorithmic transformations in certain layers of BCNN, including earlier pooling, reformulation of linear transformations, and negative skipping, are also introduced to reduce dynamic power. Implementation results show that the proposed architecture has high throughput and energy efficiency. Especially the I/O power is decreased dramatically compared with existing works. Therefore, our architecture is well suited for embedded vision-based applications that have limited power budget.

References

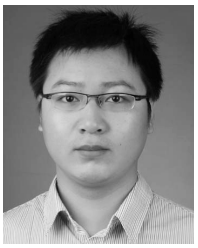
[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[2] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.
[3] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 4277–4280.
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, Sep. 2014. [Online]. Available: http://arxiv.org/abs/1409.1556
[5] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[6] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, Mar. 2016. [Online]. Available: http://arxiv.org/abs/1603.05027
[7] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *Proc. 52nd Annu. Design Autom. Conf.*, 2015, Art. no. 108.
[8] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. Int. Symp. Comput. Archit.*, 2016, pp. 367–379.
[9] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 92–104, 2015.
[10] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proc. 25th Ed. Great Lakes Symp. (VLSI)*, 2015, pp. 199–204.
[11] Y. Wang *et al.*, "Low power convolutional neural networks on a chip," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 129–132.
[12] F. Sun, J. Lin, and Z. Wang, "Intra-layer nonuniform quantization for deep convolutional neural network," in *Proc. Int. Conf. Wireless Commun. Signal Process. (WCSP)*, 2016, pp. 1–5.
[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, Oct. 2015. [Online]. Available: http://arxiv.org/abs/1510.00149
[14] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," *CoRR*, vol. abs/1602.07360, Feb. 2016. [Online]. Available: http://arxiv.org/abs/1602.07360
[15] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 1737–1746.
[16] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.
[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.
[18] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.
[19] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 236–241.

[20] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Programm. Gate Arrays*, 2017, pp. 65–74.
[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[22] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field Program. Logic Appl.*, Aug. 2009, pp. 32–37.
[23] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *Proc. IEEE CVPRW*, Jun. 2011, pp. 109–116.
[24] V. Gokhale, J. Jin, A. Dundar, and B. Martini, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jan. 2014, pp. 696–701.
[25] P. Merolla, R. Appuswamy, J. V. Arthur, S. K. Esser, and D. S. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," *CoRR*, vol. abs/1606.01981, Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.01981
[26] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2849–2858.
[27] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 4820–4828.
[28] S.-F. Hsiao, M.-R. Jiang, and J.-S. Yeh, "Design of high-speed low-power 3–2 counter and 4–2 compressor for fast multipliers," *Electron. Lett.*, vol. 34, no. 4, pp. 341–343, 1998.
[29] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010.
[30] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 8, pp. 1225–1229, Aug. 2010.
[31] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, Mar. 2004.
[32] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2013, pp. 130–137.
[33] X. Wu, Y. Wu, and Y. Zhao, "Binarized neural networks on the ImageNet classification task," *CoRR*, vol. abs/1604.03058, Apr. 2016. [Online]. Available: http://arxiv.org/abs/1604.03058
[34] H. Alemdar, N. Caldwell, V. Leroy, A. Prost-Boucle, and F. Pátrot, "Ternary neural networks for resource-efficient AI applications," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2017, pp. 2547–2554.
[35] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating deep convolutional networks using low-precision and sparsity," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2017, pp. 2861–2865.



Yizhi Wang received the B.S. degree in electronic engineering from Nanjing University, Nanjing, China, in 2016, where he is currently pursuing the M.S. degree in electronic and communications engineering.

His current research interests include VLSI architectures design and model compression algorithms for machine learning.



Jun Lin (S'14–M'16) received the B.S. degree in physics and the M.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2007 and 2010, respectively, and the Ph.D. degree in electrical engineering from Lehigh University, Bethlehem, PA, USA, in 2015.

From 2010 to 2011, he was an ASIC Design Engineer with AMD, Sunnyvale, CA, USA. In 2013, he was an Intern with Qualcomm Research, Bridgewater, NJ, USA. In 2015, he joined the School of Electronic Science and Engineering, Nanjing University, where he is currently an Associate Research Professor. His current research interest includes low-power high-speed VLSI design, specifically VLSI design for digital signal processing and cryptography.

Dr. Lin is a member of the Design and Implementation of Signal Processing Systems Technical Committee of the IEEE Signal Processing Society. He was a co-recipient of the Merit Student Paper Award at the IEEE Asia Pacific Conference on Circuits and Systems in 2008. He was a recipient of the 2014 IEEE Circuits and Systems Society Student Travel Award.

Zhongfeng Wang (F'16) received the B.E. and M.S. degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, in 2000.

He was with National Semiconductor Corporation, Santa Clara, CA, USA. He was an Assistant Professor with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA. He was a leading VLSI Architect with Broadcom Corporation, Irvine, CA, USA, for nearly nine years, where he has contributed significantly on 10 Gb/s and beyond high-speed networking products. Additionally, he has made critical contributions in designing forward error-correction codes coding schemes for 100 and 400 Gb/s Ethernet standards. His technical proposals have been adopted by many international networking standards. He joined Nanjing University, Nanjing, China, in 2016, as a Distinguished Professor through the State's 1000-Talent Plan. He is currently a World-Recognized Expert on VLSI for Signal Processing Systems. Since 2007, in the current record, he has had five papers ranked among top 20 most downloaded manuscripts in the IEEE TRANSACTIONS ON VLSI SYSTEMS. He has authored over 150 technical papers, edited 1 book (*VLSI*), and filed tens of U.S. patent applications and disclosures. His current research interests include digital communications, machine learning, and efficient VLSI implementation.

Dr. Wang has served as a Technical Program Committee Member (or Co-Chair), the Session (or Track) Chair, and a Review Committee Member for a number of international conferences. He was a recipient of the IEEE Circuits and Systems Society VLSI Transactions Best Paper Award in 2007 and the Best Paper Award Selection Committee for the IEEE Circuits and System Society in 2013. Since 2004, he has been an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS (TCAS) I, the IEEE TCAS-II, and the IEEE TRANSACTIONS ON VLSI SYSTEMS for numerous terms. He is currently a Guest Editor of a Special Issue on the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS.