An Efficient Implementation of Convolutional Neural Network With CLIP-Q Quantization on FPGA

Wei Cheng, Ing-Chao Lin, *Senior Member, IEEE*, and Yun-Yang Shih

***Abstract*— Convolutional neural networks (CNNs) have achieved tremendous success in the computer vision domain recently. The pursue for better model accuracy drives the model size and the storage requirements of CNNs as well as the computational complexity. Therefore, Compression Learning by InParallel Pruning-Quantization (CLIP-Q) was proposed to reduce a vast amount of weight storage requirements by using a few quantized segments to represent all weights in a CNN layer. Among various quantization strategies, CLIP-Q is suitable for hardware accelerators because it reduces model size significantly while maintaining the full-precision model accuracy. However, the current CLIP-Q approach did not consider the hardware characteristics and it is not straightforward when mapped to a CNN hardware accelerator. In this work, we propose a software-hardware codesign platform that includes a modified version of CLIP-Q algorithm and a hardware accelerator, which consists of 5 × 5 reconfigurable convolutional arrays with input and output channel parallelization. Additionally, the proposed CNN accelerator maintains the same accuracy of a full-precision CNN in Cifar-10 and Cifar-100 datasets.**

***Index Terms*— Convolutional neural network, CLIP-Q, accuracy, energy, hardware implementation.**

I. Introduction

In RECENT years, convolutional neural networks (CNNs) have been widely used in many applications, such as image classification [1]–[3], object detection [4]–[7], semantic segmentation, [8]–[11], visual question answering, [12]–[15], speech recognition [16], and self-driving cars [17]. CNN achieves higher model accuracy than traditional image processing methods in the above applications given enough training data.

However, to achieve better accuracy, the number of layers as well as the complexity of CNN models has increased significantly. The increased model complexity leads to an exponentially growing computational time of a CNN. For example, CNN models with more than 100 layers, such as ResNet101 [18] and DenseNet121 [19], require a considerable amount of computing resources and memory space. In order to use computing resources and memory space more efficiently, quantization, which simplifies and optimizes the CNN model, has become a popular research field.

Quantization [31], [35], [37] constrains a data representation to a smaller set, for example, using an 8-bit fixed point format to represent a 32-bit floating point format. Because fewer bits are used to represent a number, quantization greatly reduces storage requirements. For example, the authors in [20] use a 16-bit and 8-bit fixed point format to represent data. Binary neural networks (BNNs) [21], [22] and ternary neural networks (TNNs) [23] represent data in a CNN with less than two bits, which reduces the memory space requirement for more than sixteen fold.

Recently, many attempts have been made to deal with the model sparsity through model compression [35]–[37], and Compression Learning by InParallel Pruning-Quantization (CLIP-Q) has been proposed in [24], [25]. It quantizes the full-precision weights by combining pruning and weight quantization into a single learning framework during CNN model training. Weight fine-tuning is also applied after model training is completed. Full-precision weights are discarded while quantized weights are kept. There are significantly fewer quantized weights than full-precision weights since these weights are compressed and stored in a sparse encoding format. Joint pruning and quantization help CLIP-Q achieve near-zero accuracy drop compared with full-precision models.

Meanwhile, in order to accelerate CNN computation, many hardware CNN accelerators have been proposed. Instead of using a full-precision format, model weights, activation, and/or input are quantized. Because the computing units in the accelerators are designed according to the quantized data, the hardware CNN accelerator can effectively accelerate CNN computation. For example, if BNN only uses $+1$ and -1 to represent inputs and weights, XNOR gates can be used to replace the multiplication in a BNN. The authors in [26] designed a highly parallelized hardware CNN accelerator based on a BNN using XNOR for multiplication. The authors in [27] proposed a BNN accelerator, in which all convolutional operations are binarized and unified, to achieve better performance and energy efficient. However, for BNN accelerators,

Manuscript received 2 April 2022; revised 23 June 2022; accepted 9 July 2022. Date of publication 4 August 2022; date of current version 29 September 2022. This work was supported in part by the Ministry of Science and Technology under Grant 110-2221-E-006-084-MY3 and Grant 109-2628-E-006-012-MY3; and in part by the Intelligent Manufacturing Research Center from the Featured Areas Research Center Program by the Ministry of Education, Taiwan. This article was recommended by Associate Editor J. Di. *(Corresponding author: Ing-Chao Lin.)*

Wei Cheng and Ing-Chao Lin are with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan (e-mail: iclin@mail.ncku.edu.tw).

Yun-Yang Shih is now with MediaTek Inc., Hsinchu 300, Taiwan.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3193031.

Digital Object Identifier 10.1109/TCSI.2022.3193031

due to the limitation related to data precision when using these weights, these CNN hardware accelerators could not achieve the accuracy of full-precision weights.

CLIP-Q quantizes CNN weights and maintains full-precision accuracy. Meanwhile, due to high computational complexity, it is a trend to design a hardware accelerator to accelerate CNNs. However, CLIP-Q in [24], [25] did not consider the hardware characteristics, and the method used to apply CLIP-Q when designing a CNN hardware accelerator is not straightforward. In order to design a CNN accelerator with CLIP-Q, we thus propose a software-hardware codesign platform that includes both the software flow and a hardware accelerator. Based on the results obtained from the software flow, we design an efficient CNN hardware accelerator. The contributions of this paper can be summarized as follows:

- We propose a software-hardware codesign platform that includes both a software flow and a hardware accelerator. In the software flow, the parameters of CLIP-Q are determined. A CNN with the proposed CLIP-Q setup and adjustment only requires four 8-bit weights for a layer and 8 bits for activation and still has the same accuracy as full-precision CNN in Cifar-10 and Cifar-100.
- In the hardware accelerator, we propose a simple but effective weight decoder to retrieve weights during convolutional operations.
- We implement a hardware CNN accelerator with a parallel architecture and design a reconfigurable convolutional array that performs convolutional operations with various kernel sizes.
- The simulation results show that the proposed CNN hardware accelerator achieves better Giga Operations Per Second Per Watt (GOP/S/W) than the state-of-the art approach.

The rest of the paper is organized as follows: Section II introduce the background, and Section III introduces the software-hardware codesign platform and the software flow on the platform. Section IV details the architecture of the CNN hardware accelerator. Section V introduces the experimental results, and Section VI concludes the paper.

II. Background

A. CNN and Quantization NN

CNN has been widely used in many applications, including image classification [1]–[3], object detection [4]–[7], semantic segmentation [8]–[11], visual question answering [12]–[15], speech recognition [16], and self-driving cars [17]. CNN is mainly composed of convolutional, pooling and fully connected layers, as shown in Figure 1. CNN performs feature extractions through multiple convolutional layers and outperforms many current image processing methods.

However, with the increasing number of layers and model complexity in CNN, CNN requires considerable memory and computing resources. As shown in Figure 1, with regard to memory usage, the weights of each convolutional layer require $K*K*I_C*O_C*W_{size}$ bytes of memory space, where $K*K$ is the kernel size; I_C is the input channel; Oc is output channel, and W_{size} is the number of bytes for each weight.

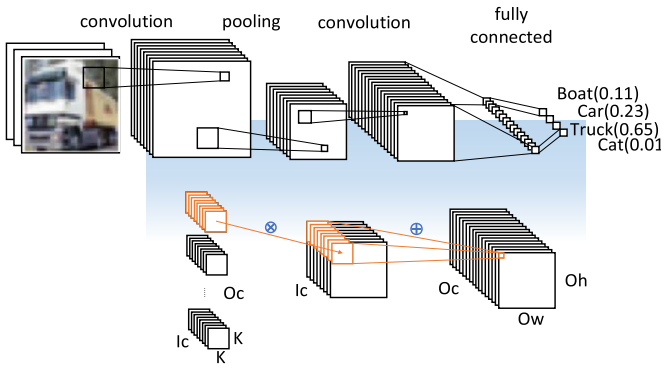

Fig. 1. CNN overview and details of a convolutional layer.

With regard to computations, if additions are not counted, each convolutional layer still requires $K*K*I_C*O_C*O_W*O_H$ multiplications, where O_W is the output width, and O_H is the output height. The larger the input and output channels, the more memory usage and computation are required, which leads to increased latency and energy consumption. Therefore, reducing memory usage and computation requirement to accelerate CNN has attracted extensive attention.

In a CNN, weight quantization is a widely used technique to reduce memory usage and computation demands. In weight quantization, weights are constrained to a set of discrete values, allowing the weights to be represented using fewer bits. Research in quantization includes [20] and [28]. These studies quantize data into 16-bit or 8-bit fixed-point formats using flexible quantization algorithms. Therefore, memory usage is only 1/2 to 1/4 that of the original size while full-precision accuracy is still maintained.

To further reduce memory usage and computation, binary neural networks (BNNs) [21] and ternary neural networks (TNNs) [23] have been proposed. BNN only uses $+1$ and -1 to represent data, while TNN uses $+1$, 0, and -1 for data representation. These methods only require 1 or 2 bits to represent a weight or input in a CNN, which can significantly reduce memory usage. However, due to the limited precision of the data representation, the accuracy of BNNs and TNNs is lower than that for a CNN with full-precision weights.

B. CLIP-Q

CLIP-Q is a CNN quantization algorithm that combines pruning and quantization into a single learning framework. The joint pruning and quantization help CLIP-Q achieve the accuracy of full-precision weights with significantly reduced memory usage. The authors in [24], [25] showed that a CNN with CLIP-Q can preserve the same accuracy as a CNN with full-precision weights.

Figure 2 shows the four steps in CLIP-Q. The first step is clipping, where weights that are close to 0 are pruned to 0. The parameter P is the proportion of the weights that are pruned to 0. In Figure 2, P is set to 0.2, indicating that 20% of the positive weights will be changed to 0, and 20% of the negative weights will be changed to 0 as well. Note that the weights that are closer to 0 are selected first. In Figure 2(b),

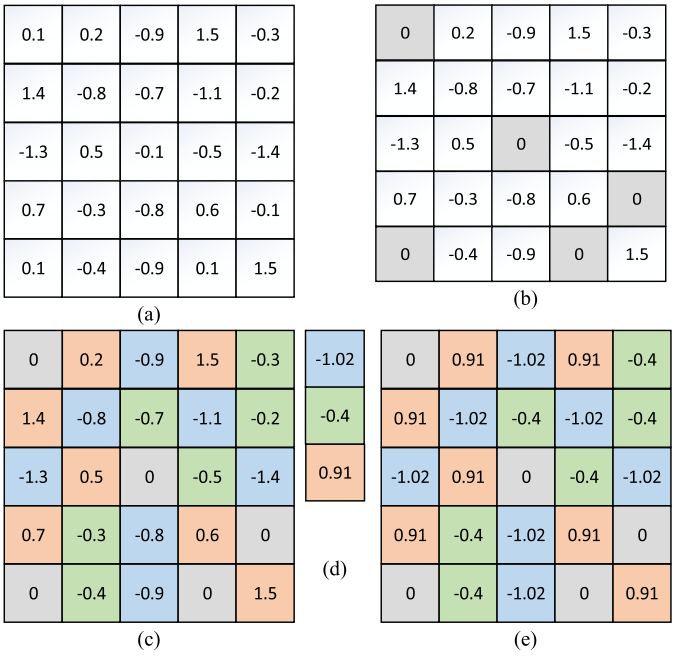

Fig. 2. An example of CLIP-Q with 25 weights, P = 0.2, B = 2 (a) Original weights (b) Clipping (c) Partitioning (d) Average (e) Quantizing.

the weights with the gray background are the weights that become 0 after clipping.

The second step is partitioning. Given a predefined number B, this step divides the remaining weights into 2^B-1 segments. In Figure 2(c), Parameter B is set to 2; hence, the remaining weights are partitioned into three segments. The partitioning method used in this work is the linear partitioning method, which is the same as in [24], [25]. However, other partitioning methods can be used to improve accuracy. The blue, green, and orange blocks in Figure 2(c) are three segments after partitioning.

The third step is averaging and quantizing. First, the average of all the numbers in each segment is computed. After that, the averages represent all the segment numbers. As shown in Figure 2(d), −1.02, −0.4, and 0.91 are the averages of these three segments, respectively. Then, the three numbers replace all the numbers in the blue, green, and orange blocks, as shown in Figure 2(e). Then, these weights are quantized. After CLIP-Q, if number 0 is counted, the weights of a CNN layer have only 2^B different numbers. Therefore, these quantized weights can be stored in an array with B-bit weight indexes, and these indexes are decoded to retrieve the quantized weight during computation.

Table I shows the network size and model accuracy of AlexNet [2], GoogLeNet [3], and ResNet50 [18] when a model is uncompressed and when a model is processed by CLIP-Q. It can be observed that the accuracy of CLIP-Q enabled models still matches the accuracy of uncompressed ones with full-precision weights. In other words, CLIP-Q dramatically reduced the storage and computational requirements with minimum overhead, and it makes CLIP-Q particularly suitable for CNN hardware accelerator designs.

Although CLIP-Q offers great model compression rate and model accuracy, the original algorithm proposed in [24], [25] did not consider the characteristics of the hardware, and it is not straightforward to map a CLIP-Q enabled model to a CNN hardware accelerator. The original CLIP-Q algorithm offers a high degree of freedom so different parameter B is used for different layers (weights in a layer will be quantized to 2^B segments), and this leads to inefficient hardware design as the accelerator will have to select weights from variant length of segments. In our design, we set the parameter B to 2 so that model weights in all layer are quantized to 2^2 segments. Also, each weight is represented by 8 bits so that the 32-bit memory bandwidth can be fully utilized by reading 4 weights in a cycle. The software flow, detailed in the next section, determines the CNN model and related parameters that will be implemented in the CNN hardware accelerator. The details of the CNN hardware accelerator are explained in Section IV.

TABLE I
NETWORK SIZE COMPARISON. CLIP-Q USES EQUAL TO OR LESS THAN 8 BITS TO REPRESENT A WEIGHT

		Accuracy	Network Size
AlexNet on ImageNet	Uncompressed	-	243.9 MB
	CLIP-Q	+0.7%	4.8 MB
GoogLeNet on ImageNet	Uncompressed	-	28.0 MB
	CLIP-Q	+0.0%	2.8 MB
ResNet-50 on ImageNet	Uncompressed	-	102.5 MB
	CLIP-Q	+0.6%	6.7 MB

III. SOFTWARE AND HARDWARE CODESIGN PLATFORM

This section first gives an overview of our software and hardware codesign platform that contains both the software flow and the hardware accelerator. Then, we present how we select the neural network model for our hardware accelerator. Finally, it describes how we determine the parameters of CLIP-Q and the bit width of the weights and activations.

A. Software and Hardware Codesign Platform Overview

Figure 3 shows the overview of this platform, which contains both a software flow and a CNN hardware accelerator. In the software flow, we first select a CNN model that is suitable for hardware implementation. After that, model information, such as the number of layers and the kernel size of each layer, are determined. Then, we set up parameters P and B of CLIP-Q, and we determine the number of bits required for each weight and activation. Finally, the model training is completed on GPU servers to obtain the quantized weights.

Notice that the software flow described in the previous paragraph is very flexible. Users can choose a preferred CNN models, suitable CLIP-Q parameters, and bit widths for quantization according to their needs. Subsequently, after a proper CNN model is selected, and the bit width for weights and activation can be determined. The proposed software flow compresses the model such that it can be easily mapped to the hardware accelerator while maintains its model accuracy. The following subsections explain the details of each step in the software flow, and Section IV details the CNN hardware accelerator.

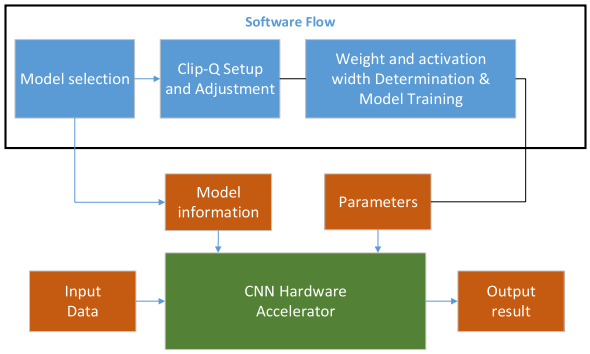

Fig. 3. Overview of software and hardware codesign platform.

TABLE II
COMPARISON BETWEEN NEURAL NETWORK MODELS. CIFAR 10 IS USED

Model	Structure	Parameter number	Accuracy
AlexNet	5 Conv. + 3 FC.	~61M	84.45%
VGG7	6 Conv. + 1 FC.	~5.5M	86.27%
GoogLenet	21 Conv. + 1 FC.	~7M	90.69%
NIN	9 Conv.	~1.2M	90.73%

B. Neural Network Model Selection

The first step is to determine a suitable CNN model for the hardware implementation. One important factor in determining a suitable model is the size of the available on-chip block RAM memory (or BRAM). Since the latency and energy required for off-chip DRAM memory access are much greater than those for on-chip memory access, if the model weights can be stored in on-chip BRAM, the accelerator will have lower latency and less energy consumption. The implementation platform used in this work was Xilinx's XC7Z0Z0 FPGA. There is only 630KB on-chip BRAM on this FPGA. When determining a model, we prefer to select a model where as many weights as possible can be stored in the on-chip memory. Note that based on user requirements, different models can be chosen.

Table II compares three model candidates: AlexNet [2], VGG7 [29], GoogLenet [3], and Network in Network [30] (NIN). The first column is the model's name, and the second column is the model structure. The third column is the number of parameters, and the fourth column is the accuracy. "Conv" stands for the convolutional layer, and "FC" stands for the fully-connected layer. As seen from Table II, NIN has the lowest number of weights and accuracy that is comparable to the other models. In addition, since there is no fully connected layer in the NIN, its structure is simpler than that of other three models. Hence, NIN was selected to be implemented in our CNN accelerator.

C. CLIP-Q Setup and Adjustment

CLIP-Q is a weight pruning and quantization technique that is able to maintain accuracy as full-precision weights while significantly reducing weight storage. Hence, it is a suitable to apply CLIP-Q on a CNN. In the first step, the CNN model is selected. In the second step, the parameters used in CLIP-Q are determined. There are two parameters: the clipping parameter P that indicates that P% of the positive weights will be clipped to 0 and the one that indicates that P% of the negative weights will be clipped to 0 as well. In this work, the clipping parameter P is set to 20 according to [24], [25] and our experimental results.

The second parameter is B, which represents that the weights in a layer are divided into 2^B segments for further averaging and quantizing. It also represents that the number of weights for a layer is 2^B. To reduce weight storage, we set B to 2 for all layers in the CNN model, which is the minimum value of B. However, the range of the weight representation can still cover positive numbers, 0, and negative numbers.

TABLE III
ACCURACY COMPARISON OF DIFFERENT BITS QUANTIZATION OF NIN

Weight bits	Cifar-10 accuracy	Cifar-100 accuracy
FULL	90.88%	67.32%
16	90.84%	67.08%
10	90.81%	67.23%
8	89.82%	65.86%
7	88.54%	61.75%
6	85.71%	60.23%
5	83.14%	59.83%
4	80.09%	57.07%

D. Weight and Activation Width Determination

After the parameters of CLIP-Q are determined, the next step is to determine how many bits are used to represent a weight and activation. In this step, we develop an in-house tool in C++ and Python to analyze the accuracy of a 9-layer NIN when different bit widths are used for the weights.

Table III shows the accuracy of the 9-layer NIN when different bit widths are used to represent a weight. The accuracy of the neural network with an 8-bit weight width is almost equal to the full precision. Hence, a weight with an 8-bit width is used in this work.

Asides from bit width, the position of the decimal point directly affects the numerical representation range and precision. Thus, we also need to choose an appropriate decimal point position.

Figure 4 shows the numerical distribution of the weights of each layer in the full-precision CNN in the NIN model. It can be seen that the distribution of the weights of the last three layers is the widest, ranging approximately between $+4$ and -4. Therefore, the appropriate weight must cover between $+4$ and -4 to cover the ranges of the weights. In addition, most of the weights in the first three layers of NIN are close to 0. In order to represent the weights of the first three layers clearly, a certain number of bits in the fraction is required to represent the weights. Based on the observation, we choose an 8-bit data format with a 3-bit integer and a 5-bit fraction for the weight. The range of the data format is $+3.96875$ to -4, which is quite close to the weight distribution, and the 5-bit fraction is adequate to represent the precision required for the weight.

Finally, we determine the data format for activations. This is important because even though the weights are quantized to 8 bits, if activations between each layer still use a full-precision 32-bit format, the accelerator still requires complex

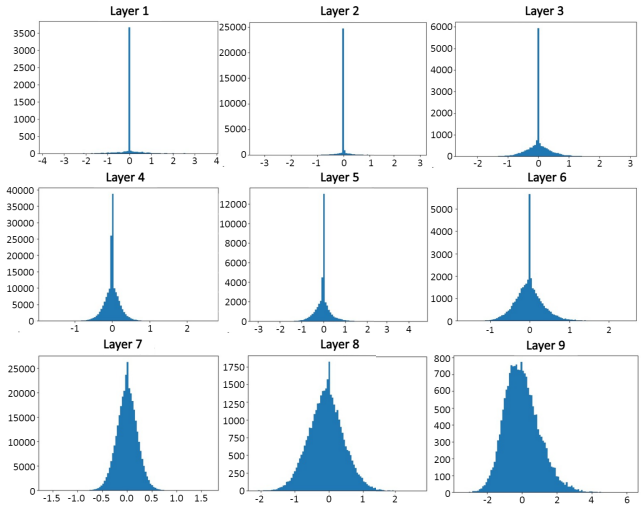

Fig. 4. Weights distribution of each layer.

TABLE IV
ACCURACY COMPARISON OF DIFFERENT FRACTION POINT POSITION OF 8-BIT DATA FORMAT IN NIN MODEL AND CLIP-Q

int. bits	frac. bits	Cifar 10	Cifar 100
>4	<4	<80%	<55%
4	4	85.53%	61.60%
3	5	90.73%	67.07%
2	6	88.09%	63.14%
1	7	87.32%	61.59%

computational circuits and a lot of memory. Therefore, it is necessary to determine the bit width and the location of the decimal point for activations such that computation and memory usage are reduced without significantly sacrificing accuracy. Since the bit width of the weights is 8, the bit width of the activations is also set to 8 to match the width of the weights. Regarding the decimal point location, we train the neural network according to the different decimal point positions to analyze the accuracy of the neural network for different decimal point positions in the activations, as shown in Table IV. From Table IV, we can see that the format with 3-bit integers and 5-bit fractions has the highest accuracy. Therefore, an 8-bit activation format that has 3-bit integers and 5-bit fractions was selected. The results are consistent with the experimental results in [30]. Therefore, it is appropriate to set the bit width of weights and activations between each layer to 8 bits with a 3-bit integer and a 5-bit fraction.

IV. CNN HARDWARE ACCELERATOR

This section first gives an overview of the CNN hardware accelerator architecture. Then, it details the design used in the accelerator to improve the parallelism. Finally, it details input and output channel parallelism used to improve the performance and the design of a reconfigurable convolutional array that performs convolutions with various kernel sizes.

A. Accelerator Overview

Figure 5 shows an overview of the accelerator architecture. There are five on-chip BRAMs in the architecture. Param

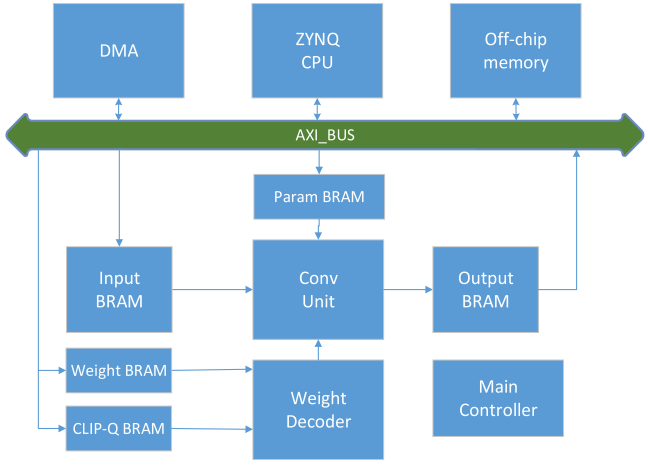

Fig. 5. CNN accelerator architecture overview.

BRAM stores various CNN parameters, including the input channel size, the output channel size, the kernel size, and the stride. Input BRAM and Output BRAM store the input and output data of a layer. Weight Index BRAM stores the 2-bit weight index for that layer. Clip-Q BRAM stores the quantized weights of each layer. The ZYNQ CPU controls the to transfer data between the off-chip DRAM memory and the on-chip BRAM through AXI protocol.

The main controller receives information from the CPU and controls the entire execution. The Conv Unit is the circuit that performs the convolutional operations. There are four convolutional modules (Conv Modules) inside the Conv Unit, and each Conv Module contains four reconfigurable convolutional arrays (RCAs). Each reconfigurable convolutional array has 25 processing elements (PEs). The Conv Unit obtains weights through the weight decoder. The design of the weight decoder and reconfigurable convolutional array are detailed in later subsections.

The execution flow of the CNN hardware accelerator is as follows: First, all input and weight data will be placed in the off-chip DRAM memory. The ZYNQ CPU controls the DMA through the AXI BUS and puts the data into the corresponding BRAM. After the data is placed, the CPU controls the controller to begin the operations. The Conv Unit performs the convolutional operations using the input data and the weights after decoding. The quantized activations are 8-bit. The output is stored in the output BRAM. After a layer finishes computing, the role of the Input BRAM and Output BRAM are exchanged. Therefore, each layer only has to read the weights of the layer from the off-chip DRAM memory. After the last convolutional layer finishes calculating, the DMA moves the results back to the off-chip DRAM memory from the output BRAM.

Section III discusses the fact that after Clip-Q setup and adjustment, the weight bit width of each layer is the same. In addition, since we limit the number of segments in weight partitioning to 4, when designing the weight decoder, we only have to use a multiplexer that has 4 inputs and a 2-bit selector to decode the weights of each layer. Figure 6 shows the design of the weight decoder, which contains four registers and a

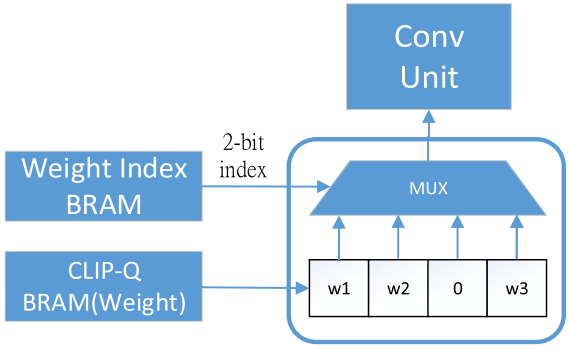

Fig. 6. Details of weights decoder.

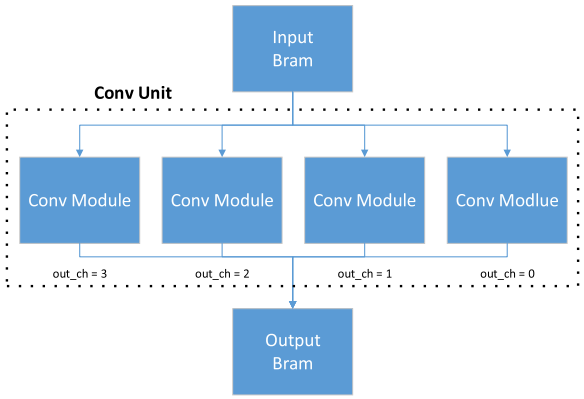

Fig. 7. Conv unit design.

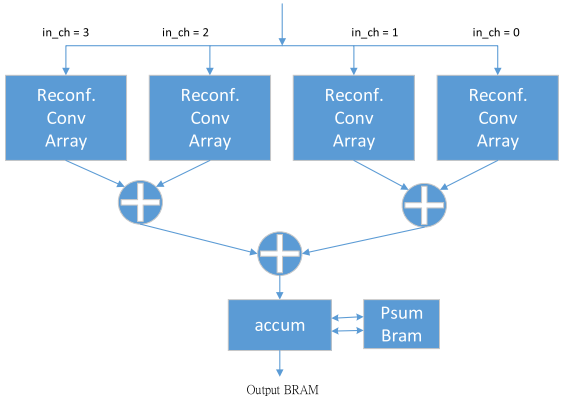

Fig. 8. Conv module with input and output channel parallelization.

multiplexer. The four registers store the four 8-bit weights of a layer. The Weight Index BRAM provides the 2-bit index of the weights. Based on the weight index, the decoder can select the proper weight that will be used in the Conv Unit. In this way, a decoder with small area can be built. Note that it is possible to have different weight bit widths for each layer. However, supporting different weight bit widths for different layers will increase the complexity of the hardware design.

B. Input and Output Channel Parallelization

In order to improve performance, our design improves the parallelization of memory access and the computational operations in both input and output channels. Figure 7 shows the detailed architecture of the Conv Unit shown in Figure 5. To speed up the operations, a parallel architecture is designed. According to the quantization results in Section III, each input data has only 8 bits. Based on the memory bandwidth, a standard BRAM has a 32-bit word. Therefore, on-chip BRAM can read or write four 8-bit data at the same time.

To make most use of the input and output memory bandwidth, our circuits are designed to improve the parallelization of the input and output channel. Figure 8 shows the Conv Module design that improves the parallelism. An input data image is considered to be 3-dimensional (3D) data because it normally has a width, a length, and channels. To improve the parallelism, the channel-major layout is used in this work to store the 3D input data stored in on-chip BRAM. Therefore, the input data of 4 channels in the same position can be retrieved in the same cycle.

In Figure 8, four convolutional computing arrays execute in parallel, for which the results are added through the adder tree. After the computational results enter the *accumulator* (accum), they are accumulated and stored in the partial sum BRAM (Psum BRAM) in the Conv Module. After the last input channel data enters the Conv Module, four 8-bit output activations are generated in parallel from the four Conv Modules and combined into a 32-bit output to be written into the Output BRAM.

Similarly, the output memory also has the same bandwidth as the input memory; that is, one cycle can write four 8-bit outputs. In addition, in the convolutional operation, the same input data are calculated with different weights to obtain the output of different channels. This is also considered in our design to improve the parallelism in the output channel. The proposed circuit is not shown but is similar to the circuit in the previous paragraph, which is implemented in four parts, corresponding to four consecutive output channels. After input data enters the circuit, four 8-bit outputs are generated from the four circuits. Finally, by directly combining the four 8-bit outputs into a 32-bit output, the output of four consecutive channels can be written to the output memory in one cycle. Therefore, the required time for the computation is only one fourth of the original time through output channel parallelization.

C. Reconfigurable Convolutional Array

Since the NIN model consist of convolutional layers with different kernel sizes, to adapt to various kernel sizes during convolutional operations, we designed a reconfigurable convolutional array that can perform convolutional operations for various kernel sizes.

Figure 9 shows the hardware architecture of the reconfigurable convolutional array. Within the reconfigurable convolutional array, there are 25 processing elements (PEs) where each PE has a multiplier, an adder, and a register. The results of multiplication and addition are saved in the register. First, each weight is stored in its corresponding PE. Then,

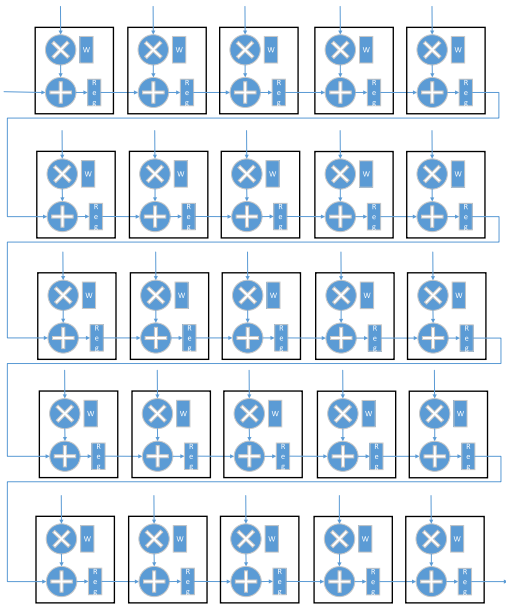

Fig. 9. Reconfigurable convolutional array. It contains 25 PEs that are connected in series.

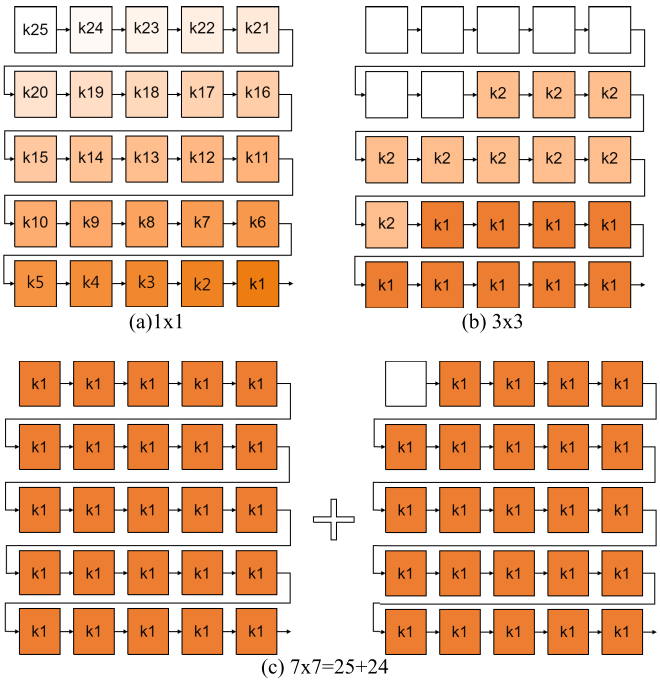

Fig. 10. Reconfigurable convolutional array for different kernel sizes.

the same input data is broadcast to each PE and multiplied by the weight stored in the corresponding PE. Finally, the multiplication results are added to the results from the previous PE and are stored in the register. This architecture reduces weight movements by reusing weights, in turn reducing energy consumption.

Figure 10 shows how the reconfigurable convolution computing array carry out convolution for various kernel sizes. When the kernel size is less than 5×5, such as 1×1 or 3×3, a kernel will only occupy the same amount of PEs of its size. For instance, each 1×1 kernel will occupy a PE in the array so in total twenty-five 1×1 kernels can be placed in the reconfigurable convolution array. As for 3×3 kernels, each kernel occupies 9 PEs and two 3×3 kernels can be placed in the array at a time. Given the kernel size equals to 5×5, it is obvious that all PEs are occupied for convolution. When the kernel size is greater than 5×5, the system completes the convolution by dividing the operations into several reconfigurable convolutional arrays, where each array performs a convolutional operation on up to 25 inputs. Take 7×7 kernel size as an example, as shown in Figure 10. The kernel can be split into two smaller kernels, whose sizes are 25 and 24, respectively. After using the two kernels to perform the convolutional operation, the results are added together to complete the convolutional operation for kernel sizes larger than 5×5. Hence, although there are only 25 PEs in the proposed reconfigurable convolutional array, the reconfigurable convolutional array can perform convolutional operations for kernel sizes larger than 5×5.

To take advantage of data reuse, the sliding window is moving downwards, so the partial sum of the previous operation can be reused. One row of the kernel data can generate a valid output. Taking a 3×3 kernel size as an example, one output is generated after every three input data values enter the array. In other words, an average of three clock cycles will have an output. According to the row size of the kernel, the number of cycles needed to generate an output can be determined.

V. Experimental Setup and Results

This section first introduces the experimental environment. Then, it details the accuracy and performance comparison.

A. Experimental Setup

The neural network is built with Python, and CLIP-Q is used to quantize model weights, where P is set to 20%, and B is set to 2. The bit width of the weights and activations are 8-bit, and the computational data are quantized to 8 bits. Therefore, during CNN inference, the data format in the software and hardware computation are equivalent, and the accuracies of the inference are also the same. After training, the parameters with the highest accuracy are saved, i.e., the model's weight and bias are saved into a file to facilitate the hardware implementation.

This design is implemented on the PYNQ-Z2 FPGA development board, where the FPGA chip is XC7Z020. The design is implemented in Verilog, and the development software is Vivado (v2018.3). The hardware resources utilized in the system are shown in Table V. Due to the limited number of DSPs, the 8-bit multipliers are synthesized using LUT. Therefore, the utilization rate of LUT is relatively high. The DSP is mainly used by the Controller circuit to calculate the data address. Compared to DSP, implementing 8-bit multipliers with LUT can reduce energy consumption. The input BRAM and output BRAM, which contain the input and output data for a layer, account for a large part of the BRAM usage.

TABLE V
RESOURCE UTILIZATION ON XC7Z020

	LUT	LUTRAM	FF	DSP	BRAM
Available	53,200	17,400	106,400	220	140
Used	42,416	521	22,498	5	120
Percent (%)	79.73	2.99	21.14	2.27	85.71

TABLE VI
ACCURACY COMPARISON OF DIFFERENT QUANTIZATION ALGORITHMS IN THE NIN MODEL

quantization	input bits	weight bits	Cifar 10	Cifar 100
Full	32	32	90.88%	67.32%
8-bit	8	8	89.82%	65.86%
TNN	2	2	83.74%	53.02%
BNN	1	1	81.52%	40.36%
8-bit CLIP-Q	8	8 (only 4 weights)	90.73%	67.07%

B. Accuracy Comparison

After quantization and the CLIP-Q fixed-segment adjustments, there are only four 8-bit weights per layer, which conserves a considerable amount of storage. However, it is important the accuracy of the model is maintained. If the accuracy can be maintained, the proposed CLIP-Q is suitable for quantization implemented in the CNN accelerator. Table VI compares the Cifar-10 and Cifar-100 accuracy of the models using different quantization methods. All the accuracies in Table VI are generated by experiments in the 9-layer NIN model. FULL means that the 32-bit float full precision data format is used. 8-bit represents the input, and weight data are quantized to 8-bit precision. TNN means that all data are represented by $+1$, 0, -1, and BNN only uses $+1$ and -1 to represent data.

It can be seen from Table VI that although TNN and BNN save a lot of storage space for weights and inputs, the accuracy is reduced. Especially in the Cifar-100, the accuracy is significantly different from the full precision. The accuracy of the 8-bit precision is closer to the full precision, but there is still a 2% drop in the Cifar-100 test data. However, the adjusted 8-bit CLIP-Q has almost the same accuracy as the full-precision, and only four 8-bit weights are needed. The proposed CLIP-Q significantly reduces storage space and can achieve almost the same accuracy as full precision.

C. Performance Comparison

Table VII shows the required cycles to read input feature maps for various kernel sizes in the reconfigurable convolution design. In the proposed 5×5 reconfigurable convolutional array, as long as the kernel size is not higher than $25(= 5 \times 5)$, it is only necessary to read the input feature map once to complete the convolution. For convolution kernel sizes higher than 5×5, the kernel is divided into smaller kernels, each of which is equal to or smaller than 25. Therefore, the input access time of large-size kernels is equal to the kernel size divided by 25. Compared to [32], where convolutions of different kernel sizes were completed using a 3×3 kernel

TABLE VII
COMPARISON OF INPUT FEATURE MAP READING TIMES IN DIFFERENT KERNEL SIZE

kernel size	[32]	ours
1x1	1	1
3x3	1	1
5x5	4	1
7x7	9	2
9x9	9	4
11x11	16	5

TABLE VIII
COMPARISON BETWEEN DIFFERENT IMPLEMENTATIONS ON FPGA

	[33]	[34]	[32]	ours
Platform	Virtex 7 VX485t	Virtex 7 VX485t	Zynq XC7Z020	Zynq XC7Z020
Clock	100Mhz	156Mhz	214Mhz	111Mhz
Data format	32-bit float	16-bit fixed	8-bit fixed	8-bit fixed
Power(W)	18.61	30.2	3.5	2.147
GOP/S	61.62	565.9	84.3	91.1
GOP/S/W	3.31	22.15	24.1	42.4

size, our reconfigurable design reduces the input data access time. Thus, it can also complete convolution faster than was the case in [32].

Table VIII shows a comparison between the proposed design and related work. Because the proposed 5×5 convolutional array improves the input and output channel parallelism, the overall GOP/S performance was increased. Furthermore, since the multipliers were synthesized using LUT, the power consumption was less than when using DSP. Also, we used fewer flip-flops on the convolution circuit, which also reduced the power consumption. According to Table VIII, our CNN accelerator's GOP/S/W were relatively higher than those in other work, which means the proposed design had higher energy efficiency.

VI. RELATED WORK

In this section, we discuss previously designed FPGA-based CNN accelerator. Angel-Eye [32] proposed a software-hardware codesign for embedded CNN applications and used a 3×3 convolver to handle different computational workloads of various kernel size; however, the utilization rate of its 3×3 convolvers is only 1/9 when dealing with 1×1 kernel. Instead of designing the accelerator directly. High-level synthesis was used to generate the design with the help of roofline model in [33]. It measured the compute and memory requirements for each layer of a CNN model and came up with suitable architectures that efficiently utilize the memory bandwidth. However, it mapped full precision CNN models directly to FPGAs without considering the underlying hardware costs, and common strategies such as data quantization and model pruning were not applied. An end-to-end FPGA-based CNN accelerator aiming for high throughput and high resource

utilization was proposed in [34]. While different layers have different compute-to-memory ratio, it proposed a batch-based method for fully connected layer to better utilize memory bandwidth. It adopted 16-bit data quantization for input and weight data; however, its models were not pruned and was unfriendly to resource-limited FPGAs.

VII. Conclusion

CLIP-Q significantly reduces the CNN weight storage requirement while also maintaining accuracy. This feature makes CLIP-Q suitable for a CNN. However, the current CLIP-Q approach did not consider the hardware characteristics and the method for applying CLIP-Q when designing a CNN hardware accelerator was not straightforward. In this work, we propose a software-hardware codesign platform that includes both the software flow and the hardware accelerator. The software flow obtained neural model parameters suitable for hardware implementation. We also designed a CNN hardware accelerator. The accelerator executed convolutions with various kernel sizes through 5×5 reconfigurable convolutional arrays and improved parallelism in both the input and output channels. The experimental results show that the proposed CNN accelerator has higher energy efficiency than the state-of-the art alternatives.

References

[1] Y. Wei *et al.*, "HCP: A flexible CNN framework for multi-label image classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 9, pp. 1901–1907, Jun. 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[3] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1–9.

[4] S. Gidaris and N. Komodakis, "Object detection via a multi-region and semantic segmentation-aware CNN model," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1134–1142.

[5] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 379–387.

[6] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[8] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," 2014, *arXiv:1412.7062*.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[10] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, "Full-resolution residual networks for semantic segmentation in street scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 4151–4160.

[11] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, *arXiv:1511.07122*.

[12] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, "Learning to reason: End-to-end module networks for visual question answering," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 804–813.

[13] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A neural-based approach to answering questions about images," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 1–9.

[14] H. Noh, P. H. Seo, and B. Han, "Image question answering using convolutional neural network with dynamic parameter prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 30–38.

[15] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, "Stacked attention networks for image question answering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 21–29.

[16] D. Palaz *et al.*, "Analysis of CNN-based speech recognition system using raw speech as input," Idiap, Martigny, Switzerland, Tech. Rep. Idiap-RR-23-2015, 2015.

[17] M. Bojarski *et al.*, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE CVPR*, Jul. 2017, pp. 4700–4708.

[20] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (ACM)*, 2016, pp. 26–35.

[21] 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," 2016, *arXiv:1602.02830*.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4114–4122.

[23] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*.

[24] F. Tung and G. Mori, "CLIP-Q: Deep network compression learning by in-parallel pruning-quantization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7873–7882.

[25] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 3, pp. 568–579, Mar. 2018.

[26] H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2017, pp. 98–105.

[27] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, and D. Wang, "FBNA: A fully binarized neural network accelerator," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 51–513.

[28] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[30] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*.

[31] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021.

[32] K. Guo *et al.*, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2017.

[33] C. Zhang *et al.*, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, vol. 2015, pp. 161–170.

[34] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–9.

[35] Z. Song *et al.*, "DRQ: Dynamic region-based quantization for deep neural network acceleration," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 1010–1021, doi: 10.1109/ISCA45697.2020.00086.

[36] X. Zhou *et al.*, "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 15–28, doi: 10.1109/MICRO.2018.00011.

[37] S. Q. Zhang, B. McDanel, H. T. Kung, and X. Dong, "Training for multi-resolution inference using reusable quantization terms," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2021, pp. 845–860, doi: 10.1145/3445814.3446741.

Wei Cheng received the B.E. degree in computer engineering from The University of Hong Kong in 2018. He is currently pursuing the master's degree with the Department of Computer Science and Information Engineering, National Cheng Kung University. His research interests lie in the field of very large-scale integration design, computer architecture, and deep neural network accelerators.

Ing-Chao Lin (Senior Member, IEEE) received the M.S. degree in computer science from the National Taiwan University, Taipei, Taiwan, and the Ph.D. degree from the Computer Science and Engineering Department, The Pennsylvania State University, State College, PA, USA, in 2007. From 2007 to 2009, he was with Real Intent Inc., Sunnyvale, CA, USA. Since 2009, he has been with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, where he is currently a Full Professor. He was a Visiting Scholar at the University of California, Santa Barbara, in 2015; and he was a Visiting Scholar at Academia Sinica in 2017. His current research interests include very large-scale integration design and computer-aided design for nanoscale silicon, energy-efficient reliable system design, and computer architecture. He has served on the technical program committee for several conferences, such as ASP-DAC, ICCAD, ICCD, and GLSVLSI. He has been an ACM Senior Member since May 2016. He was awarded the Excellent Young Researcher by Chinese Institute of Electrical Engineering in 2015, the Best Young Professionals (Formerly GOLD) by IEEE Tainan Section in 2016, and the Humboldt Fellowship for Experienced Researcher in 2019.

Yun-Yang Shih received the M.S. degree in computer science and information engineering from the National Cheng Kung University in 2020. He is currently with Mediatek Inc. His research interests lie in the field of very large-scale integration design and deep neural network accelerators.