On-Chip Memory Based Binarized Convolutional Deep Neural Network Applying Batch Normalization Free Technique on an FPGA

Conference Paper · May 2017

DOI: 10.1109/IPDPSW.2017.95

2 authors, including:

Hiroki Nakahara
Tokyo Institute of Technology
104 PUBLICATIONS **1,029** CITATIONS

On-chip Memory Based Binarized Convolutional Deep Neural Network Applying Batch Normalization Free Technique on an FPGA

Haruyoshi Yonekawa
Tokyo Institute
of Technology, Japan

Hiroki Nakahara
Tokyo Institute
of Technology, Japan

***Abstract*—A pre-trained convolutional deep neural network (CNN) is a feed-forward computation perspective, which is widely used for the embedded systems, requires highly power-and-area efficiency. This paper proposes a binarized CNN on an FPGA which treats only binary 2-values (+1/-1) for the inputs and the weights. In this case, the multiplier is replaced into an XNOR circuit instead of a dedicated DSP block. For hardware implementation, using binarized inputs and weights is more suitable. However, the binarized CNN requires the batch normalization techniques to retain the classification accuracy. In that case, the additional multiplication and addition require extra hardware, also, the memory access for its parameters reduces system performance. In this paper, we propose a batch normalization free binarized CNN which is mathematically equivalent to one using batch normalization. The proposed CNN treats the binarized inputs and weights with the integer bias. We implemented the VGG-16 benchmark CNN on the Xilinx Inc. Zynq UltraScale+ MPSoC zcu102 evaluation board. Our binarized CNN stores all the weights, inputs, and output to on-chip BRAMs those are faster and dissipate lower power than an off-chip memory, such as a DDR4SDRAM. Compared with the conventional FPGA realizations, although the classification accuracy is 6.5% decayed, the performance is 2.45 times faster, the power efficiency is slightly better, and the area efficiency is 2.68 times better. Compared with the ARM Cortex-A57, it is 136.8 times faster, it dissipates 3.1 times much power, and its performance per power efficiency is 44.7 times better. Also, compared with the Maxwell embedded GPU, it is 4.9 times faster, it dissipates 1.3 times much power, and its performance per power efficiency is 3.8 times better. Thus, our method is suitable for the embedded computer system.**

I. Introduction

A. Convolutional Deep Neural Network (CNN)

Recently, for embedded computer vision systems, a convolutional deep neural network (CNN), which consists of the 2D convolutional layers and a deep neural network, is widely used. Since the CNN emulates the human vision, it has a high accuracy for an image recognition. For example, a human face recognition [30], a human and object detection [14], a human pose estimation [33], a string recognition in a scene [16], a road traffic sign recognition [4], a sport scene recognition [18], a human action recognitions [17], [10], are reported. These researches showed that the CNN outperformed conventional techniques.

With the increase of the number of layers, the CNN can increase classification accuracy. Thus, a large-scale CNN is desired. To keep up with the real-time requirement of the embedded vision system, since the existing system using a CPU is too slow, the acceleration of the CNN is necessary [19]. Most software-based CNNs use a GPU [1], [3], [7], [31], [32]. Unfortunately, since the GPU consumes much power, they are unsuitable for the embedded system [11]. Thus, FPGA-based CNNs are required for a low-power and a real-time embedded system. As for the classification accuracy, the CNN using a fixed-point representation has almost the same accuracy as one using a floating-point representation [12]. The FPGA can use a minimum precision which reduces the hardware resources and increases the clock frequency, while the GPU cannot do it. Another merit of the FPGA is a lower power consumption than the GPU. A previous work [11] reported that, as for the performance per power, the FPGA-based CNN is about 10 times more efficient than the GPU-based one. For FPGA realizations, many implementations have been reported. For example, a hand-written character recognition [19], a face detector [28], a scene determination [26], and an object recognition [12] are implemented.

For an embedded system, it requires a high performance computation under a limited power consumption. Even if we use a mobile GPU, such as NVidia Jetson TX1, it dissipates much power. Thus, the GPU based realization is not suitable to the embedded system. In this paper, we propose the FPGA based system.

B. Binarized CNN on an FPGA

Recently, Courbariaux et al. proposed the binarized CNN, which restricts weights and inputs to 2-valued (-1 and 1) [8]. For hardware implementation, binarized inputs and weights are more suitable. The evaluation results [24] reported that the binarized CNN can deliver orders of magnitude improvements in performance and performance/watt over well optimized software on CPU and GPU. Although FPGA is less efficient than ASIC, the FPGA-ASIC gap may be reduced for designs that heavily utilize hard blocks. Hence, FPGA offers an attractive solution, which deliver superior efficiency improvements over software, without having to lock into a fixed ASIC solution. Recently, many FPGA implementations of the binarized CNN have been reported [20], [22], [23], [24], [34], [39].

However, the binarized CNN uses the batch normalization technique [29] to retain the classification accuracy [8]. In that case, the additional multiplication and addition require

more hardware, while the memory access for its parameters reduces system performance and increases power consumption. In this paper, we propose the batch normalization free CNN which is mathematically equivalent to the CNN using batch normalization. The proposed CNN consists of the binarized inputs and weights with the integer bias. Thus, the proposed one is smaller and faster than the conventional ones. The other existing problem was that the previous binarized implementations [24], [34] only evaluated a tiny CNN for the CIFAR10 task. In this paper, we show that the recognition accuracy for a large VGG-16 CNN for the ImageNet task, which is widely used for the modern CNN.

C. Proposed Method

Almost CNN implementations use high-end FPGAs and off-chip memories. Since 2D convolution operations on the CNN occupy more than 90% of computation time [6], they used many DSP blocks and consumes much power. We use the binarized CNN [8], which restricts the internal values to 2-valued (-1 and 1). In this case, our implementation uses XNOR gates instead of the DSP blocks. Thus, the power consumption is lower than the DSP block-based one. Previous work [8] showed that a classification error rate of the binarized CNN trained on the small task, which was CIFAR-10. Since the binarized one uses the batch normalization which is a mathematical trick to adjust the deviation of the biased data, it achieved near state-of-the-art CNN error rate with only a single bit per weights and signed activation functions. Thus, the binarized CNN implementation would consume less FPGA resources than the conventional one. However, since the conventional binarized one requires additional hardware and more memory access for the batch normalization, it is undesirable for a compact realization. In this paper, we propose a batch normalization free CNN which is a mathematically equivalent to the CNN using the batch normalization. It treats the binarized inputs and weights with the integer bias.

D. Contributions of the Paper

Contributions of our paper are as follows:

1. We proposed a batch normalization free binarized CNN on the FPGA. As far as we know, this is the first implementation. Our implementation replaces an excessively large number of DSP blocks into XNOR gates with near state-of-the-art CNN classification error rate.
2. The existing problem was that the previous implementations [24], [34] only evaluated the tiny CNN for the CIFAR10 task. In this paper, we showed that the recognition accuracy for the large VGG16 for the ImageNet task, which is widely used for the modern CNN.
3. We implemented the batch normalization free binarized CNN for the VGG-16 benchmark on the Xilinx Inc. Zynq UltraScale+ MPSoC zcu102 evaluation board. Compared with the recent FPGA realization [35], although the binarized CNN is 6.5% worse of the classification accuracy than the 32 bit floating one, the performance was 2.45 times faster, the performance per power efficiency was slightly better, and the performance per area efficiency was 2.68 times better.
4. We compared with the embedded CPU and the embedded GPU with respect to the VGG-16 CNN (batch size was 1). Compared with the ARM Cortex-A57, it was 136.8 times faster, it dissipated 3.1 times much power, and its performance per power efficiency was 44.7 times better. Also, compared with the Maxwell embedded GPU, it was 4.9 times faster, it dissipated 1.3 times much power, and its performance per power efficiency was 3.8 times better. We showed that the FPGA based system is suitable for the embedded computer vision system.

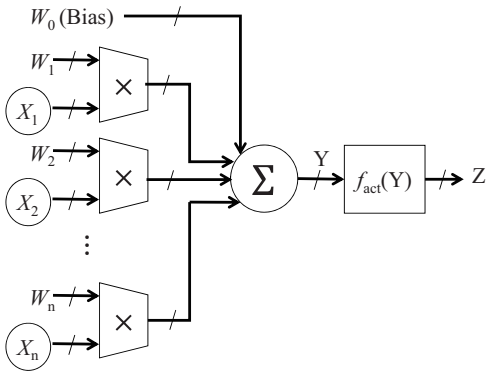

Fig. 1. Circuit for an artificial neuron (AN).

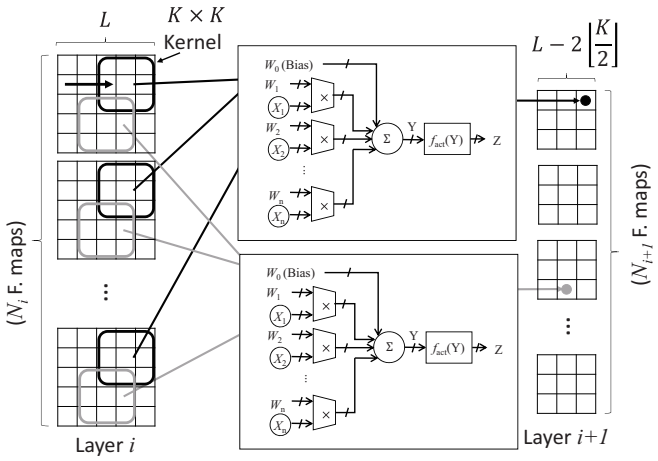

Fig. 2. 2D convolutional operation.

E. Organization of the Paper

The rest of the paper is organized as follows: Chapter 2 introduces the convolutional deep neural network (CNN); Chapter 3 introduces the binarized CNN; Chapter 4 shows the batch normalization free binarized CNN; Chapter 5 shows the experimental results; and Chapter 6 concludes the paper.

II. Convolutional Deep Neural Network (CNN)

A. Artificial Neural Network

Let n be a bit precision, $X = (x_0, x_1, \ldots, x_n)$ be the input, $Y = (y_0, y_1, \ldots, y_n)$ be the internal variable, $W = (w_0, w_1, \ldots, w_n)$ be the weight, f_{act} be the activation function,

TABLE I. SPECIFICATIONS FOR THE VGG-16 [27].

Layer	L	K	#F. maps	Repeats
Input	224	1	3	1
Conv1	224	3	64	2
Conv2	112	3	128	2
Conv3	56	3	256	3
Conv4	28	3	512	3
Conv5	14	3	512	3
FC1	1	1	4096	1
FC2	1	1	4096	1
FC3	1	1	1000	1

and $Z = (z_0, z_1, \ldots, z_n)$ be the the output. Note that, in this paper, a capital letter denotes an integer, while a small letter denotes a binary value. Fig. 1 shows a circuit for **an artificial neuron (AN)**. The following expression shows an operation for the AN:

$$\begin{aligned} Y &= \sum_{i=0}^{n} W_i X_i, \\ Z &= f_{act}(Y), \end{aligned} \tag{1}$$

where X_0 is a constant one and W_0 denotes **a bias** which corrects the deviation of the given data. Typically, the activation function is realized by a sigmoid, a tanh, a ReLU [21], and so on. **A deep neural network (DNN)** consists of multiple of ANs. **A convolutional deep neural network (CNN)** has multiple **layers**. The typical layer consists of **a 2D convolutional layer**, **a pooling layer**, and **a classification layer**. Each layer consists of multiple **feature maps**. To recognize the input image, first, the feature map reacts corresponding subdivided training data by 2D convolutional layers with pooling layers. Then, the classifier selects the appropriate reactions from feature maps. Usually, the classifier is realized by the fully connected neural network. In this paper, for layer i, K_i denotes the kernel size, N_i denotes the number of feature maps, and L_i denotes the feature map size. Fig. 2 shows the 2D convolution operation. It computes the output by shifting a $K \times K$ size **kernel**, and applies the MAC operation similarly to the AN. For (x, y) at the output feature map value $i + 1$, the following MAC (multiply-accumulation) operation is performed:

$$\begin{aligned} Y_{i+1,x,y} &= \sum_{k=0}^{N_i-1} \Big(\sum_{m=0}^{K-1} \sum_{n=0}^{K-1} X_{k,x+m,y+n} W_{k,m,n} \Big) \\ Z_{i+1,x,y} &= f_{act}(Y_{i+1,x,y}). \end{aligned} \tag{2}$$

In the 2D convolutional operation, Z is mapped to (x, y) at the output feature map $i + 1$. In the fully connected layer, $L_i = 1$ and $K_i = 1$. By inserting the non-linear and low-imaging operations into the convolution layers, we can reduce the number of computations in the convolution layers, while we can obtain the movement invariance. We call this **a pooling operation**, which can be realized by a simple circuit. In this paper, we implement the max-pooling operation. Its operation can be realized by a comparator for selecting the maximum value in the kernel. It is much smaller than the 2D convolution operation circuit.

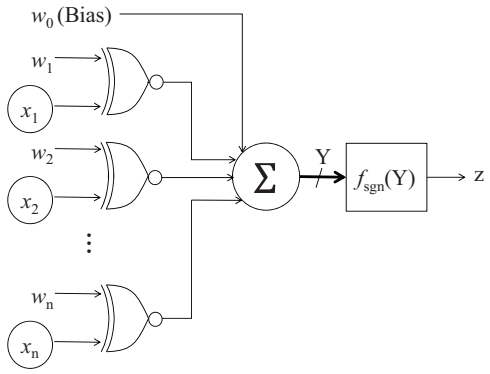

Fig. 3. Binarized AN.

TABLE II. TRUTH TABLE FOR A BINARIZED MULTIPLICATION $c = a \times b$.

a	b	c
-1	-1	+1
-1	+1	-1
+1	-1	-1
+1	+1	+1

B. VGG-16 CNN

Table I shows specifications for the VGG-16 benchmark CNN [27], which is widely used in the computer vision system. The VGG-16 has 16 layers. The basic layers consist of multiple 2D convolution layers with $K = 3$ and max-pooling layers, while the rear layers consist of fully connected neural networks. First, it receives a normalized 224×224 image, which consists of 8-bit RGB color data.

III. BINARIZED CNN

As shown in Fig. 2, the 2D convolutional operation consisting of the AN operation and the memory access for the feature maps. In the 2D convolutional operation, the MAC operation is a dominant of computation time. To focus on the efficient realization of the MAC operation, in this part, we discuss the binarization of the AN operation.

A. Definition

As shown in Exprs (1) and (2), the 2D convolutional operation is an extension of the AN operation. In this section, to briefly explain, we introduce the binarization of the AN shown in Fig. 1. Recently, Courbariaux et al. proposed **the binarized CNN** [8], which restricts the weights and inputs to only -1 and 1. The binarized AN operation performs the following expression:

$$\begin{aligned} Y &= \sum_{i=0}^{n} w_i x_i, \\ z &= f_{sgn}(Y), \end{aligned} \tag{3}$$

where $f_{sgn}(Y)$ denotes the signed activation function as follows:

$$f_{sgn}(Y) = \begin{cases} 1 & (if\ Y \geq 0) \\ -1 & (otherwise) \end{cases}$$

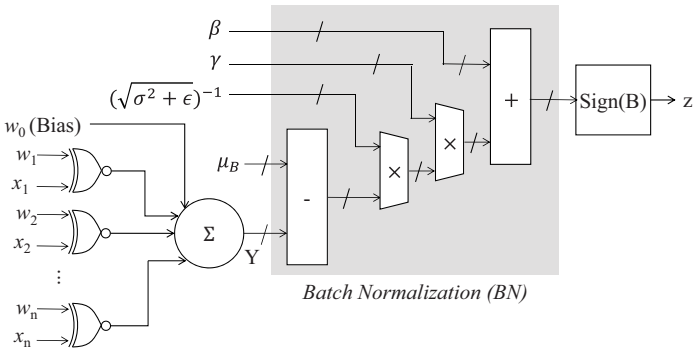

Fig. 4. Binarized AN with batch normalization (BN).

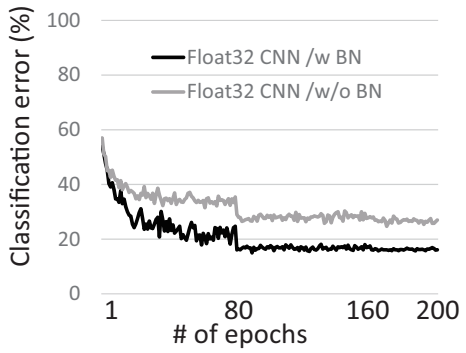

Fig. 5. Classification error for VGG-16 using the float32 bit CNN.

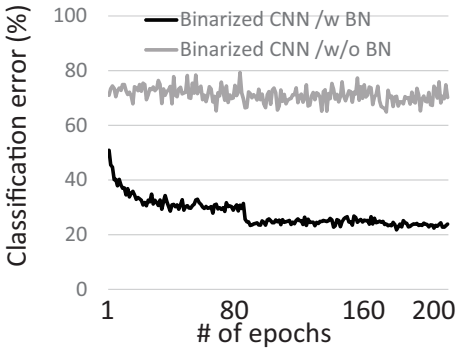

Fig. 6. Classification error for VGG-16 using the binarized CNN.

In the first layer, since the input is the RGB color images, the bit precision for the input is 8 bits. Note that, the binarized AN, the bit precision of internal variable would not be a binarized. Table II shows the binarized multiplication. Since the FPGA cannot directly represent -1, we assign a logical zero to -1. In this case, since the binarized multiplication can be realized by an XNOR gate, the hardware area for the integer multiplier is drastically reduced. Also, the other advantage is that the binarized weight can reduce the memory bandwidth than the integer one. Above two reasons, although the binarized CNN is area and performance efficient one, it requires the additional hardware for batch normalization, which is a mathematical trick to retain the classification accuracy.

B. Batch Normalization

Typically, to accelerate training time and convergence, a set of training data (mini-batch) is back-propagated and it updates weights at a time. It is called by a mini-batch training. In this case, since the impact on the difference in the distribution of data for each batch (internal covariate shift), the convergence of the training tends to be slow, and the trainer must carefully determine the initial value of the parameters. These problems are solved by **batch normalization (BN)** [29] which corrects the difference in the distribution by a shift and a scaling operations.

At the training, the BN finds parameters γ and β in order to regularize the variance to 1 and the average to 0 for each mini-batch. The BN algorithm for training is shown as follows:

Algorithm 3.1: Input: Mini-batch $B = \{X_1, X_2, \ldots, X_m\}$ Output: Parameters γ and β.

1. Obtain average for each mini-batch: $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} X_i$.
2. Obtain variance for each mini-batch: $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu_B)^2$.
3. Perform normalization: $\hat{X}_i \leftarrow \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$.
4. Obtain γ and β, that regularizes the variance to 1 and the average to 0 for $B_i \leftarrow \gamma\hat{X}_i + \beta$.
5. Terminate.

Above Algorithm performs the normalization for each mini-batch. A hyper parameter ϵ is set for a coefficient stabilization, which is used to adjust the training time. Since both γ and β have been already trained during classification, the AN with BN just reads them from the off-chip memory. Thus, from Expr. (3), the operation for the AN with BN is introduced as follows:

$$\begin{aligned} Y &= \sum_{i=0}^{n} w_i x_i, \\ B &= \gamma Y + \beta, \\ z &= f_{sgn}(B), \end{aligned}$$

where w_i, x_i, z are binary variables, while Y, B, γ, and β are integer ones. Above expression shows that the AN with BN requires additional cost for the multiplier and the adder for area, while the memory access for γ and β. Fig. 4 shows the binarized AN with BN.

C. Comparison of Classification Errors

Especially, since the binarized CNN without the BN only handles +1 and -1, the average output value is unbalanced. As a result, the activation would be also unbalanced. Thus, the classification accuracy tend to be worse compared with the float32 bit precision CNN. By introducing a batch normalization, the mean value for the internal variable in the binarized CNN can be approximated to zero. Therefore, the classification error can be improved.

As for the classification errors, we compared with the binarized CNN with the float32 bit precision one. We used the ImageNet dataset, and designed the VGG-16 CNN by the Chainer deep neural network framework [3]. Note that, both

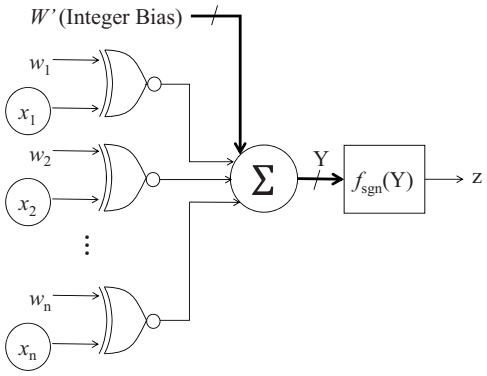

Fig. 7. BN free binarized AN.

CNNs used the same layer structure shown in Table I. To improve the classification error, we used following tricks.

1. Augmentation: At the training, we used two techniques for the data augmentation. That is, we sliding the image to up, down, left, or right in a random manner in the range from -4 to +4 pixels. The other is performed horizontally inverted at a random. However, these augmentation data are not used during the classification.
2. Normalization: We subjected to contrast adjustment for training dataset. It is a kind of input data normalization.
3. Dropout: We attached the dropout layer to the output of fully connection layers in order to inhibited overfitting.
4. Scheduling: We reduced the training rate after a certain number of epochs (in other word, an epoch means the number of trainings).

At the training, we used the ADAM training rule [9] which may reduce the impact of the weights scale. We set the number of mini-batch to 100, and the number of epochs to 200. we used the training rate scheduling to 1/10 every 80 epochs. Fig. 5 compares the classification error for the float32 bit precision CNN, while Fig. 6 compares that for the binarized one with/without the BN. and that for the binarized one with the BN or without one. From Figs. 5 and 6, the BN improves the classification error. Especially, that for the binarized CNN is drastically improved. Thus, the BN is an essential techniques for the binarized CNN. Compared with the binarized CNN using the BN with the float32 one, the difference of the classification error rate is 6.5%. Therefore, as for the classification accuracy, the binarized CNN is considerable.

IV. Batch Normalization Free Binarized CNN

A. Equivalence to Batch Normalization

Although the BN is an essential techniques for the binarized CNN, it requires additional area for the multiplier and the adder, and the memory access for parameters. In this paper, we introduce **a batch normalization free binarized CNN** which has an equivalence to the BN operation. Since it does not requires additional area, the hardware becomes simpler than the one including BN operation.

As shown in Algorithm 3.1, the BN normalizes the internal variables Y. Let Y' be the output of the BN operation. Then, we have

$$\begin{aligned} Y' &= \gamma\frac{Y-\mu_B}{\sqrt{\sigma_B^2+\epsilon}}+\beta \\ &= \frac{\gamma}{\sqrt{\sigma_B^2+\epsilon}}\left(Y-\left(\mu_B-\frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta\right)\right). \end{aligned}$$

From above expression, the signed activation function becomes

$$f'_{sgn}(Y)=\begin{cases} 1 & \left(if\ Y<-\mu_B+\frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta\right) \\ -1 & (otherwise) \end{cases}$$

That is, the value of the active function is determined by the value of the above equation. In this case, since $x_0=1$, Y can be equivalent to the following expression:

$$\begin{aligned} Y &= \sum_{i=0}^{n} w_i x_i - \mu_B + \frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta \\ &= \sum_{i=1}^{n} w_i x_i + \left(w_0 - \mu_B + \frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta\right) \\ &= \sum_{i=1}^{n} w_i x_i + W'. \end{aligned} \tag{4}$$

Thus, from Expr. (4), the binarized AN with the BN is converted into the binarized AN with an integer bias W'. In this paper, we call this the BN free binarized AN. This expression approximates the BN operation by a shifting one. Fig. 7 shows the BN free binarized AN. In the circuit, since the BN operation is replaced into the integer bias, the hardware becomes simpler and the memory access is reduced. Therefore, the BN free binarized CNN is faster and smaller than conventional ones.

V. BN Free Binarized CNN Architecture using On-chip Memory

A. Pipelined Binary 2D Convolutional Circuit

Fig. 8 shows the pipelined binary 2D convolutional circuit. To increase the throughput, we use the shift register to make a streaming data flow from the memory for the feature map. The

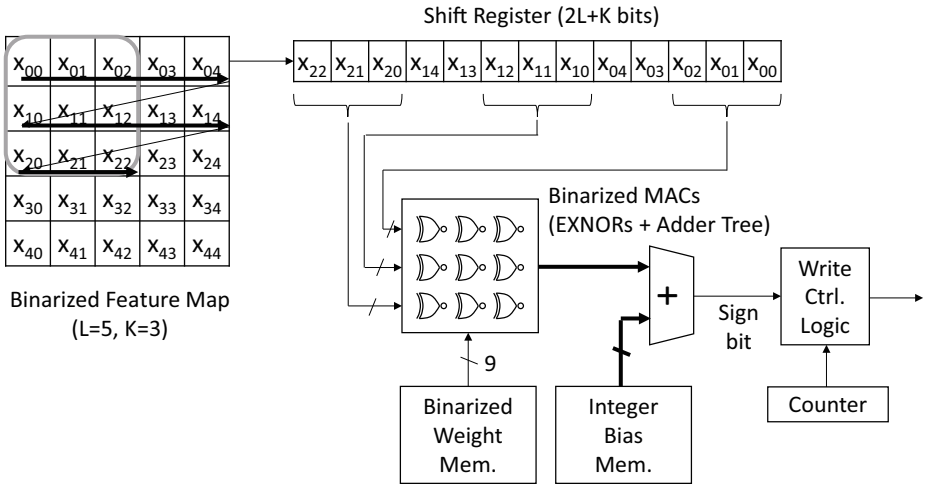

Fig. 8. Pipelined Binary 2D Convolutional Circuit.

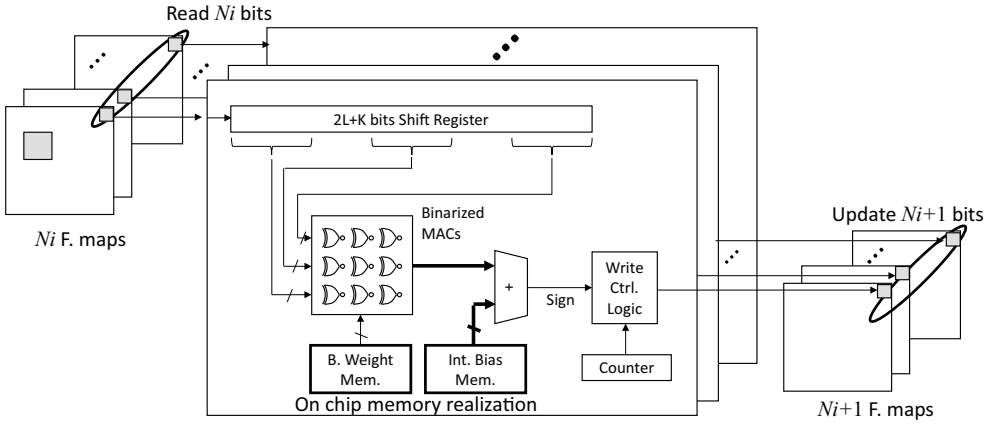

Fig. 9. Parallel Pipelined Binary 2D Convolutional Circuit.

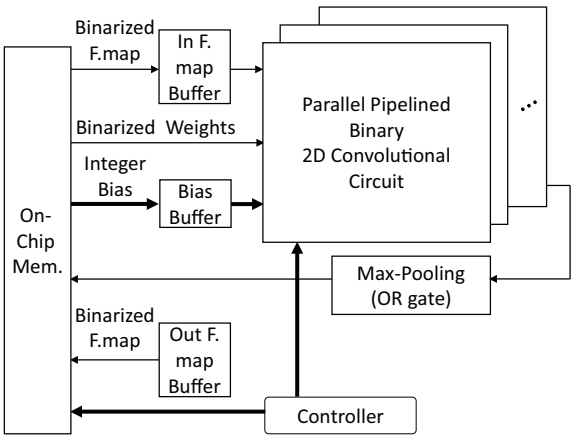

Fig. 10. Overall Architecture.

circuit read the corresponding inputs from the shift register, then it applies to the binarized MAC operation which is realized by XNOR circuits and an adder tree. Next, it adds the pre-computed bias, which is obtained by Expr. (4). Since the kernel crosses the boundary of the feature map, we attach the write control logic to the output of the circuit.

To further increase the performance, we propose the parallel pipelined binary 2D convolutional circuit shown in Fig. 9. To flexibility access by the convolutional operation, multiple BRAMs are used to realize multi-port with wide band memory access speed. It accesses multiple feature maps at a time by using the on-chip memory on the FPGA. Since we use the binarized CNN, the memory size is drastically reduced compared with non-binarized one.

B. Overall Architecture

Fig. 10 shows the architecture for the proposed BN free binarized CNN. The memory access circuit is almost the same as the conventional one, however, the memory part is realized by the on-chip block RAMs (BRAM). The proposed architecture has the shift registers and buffers to access indices for the corresponding kernel. This structure is similar to the previous architecture [35]. The difference part is that both the 2D convolutional operation and the fully connected one are binarized, and the only biases are represented by integers.

In our implementation, we realize the binarized multiplier by an XOR gate with a NOT gate on a carry-chain on the FPGA [37]. Thus, our implementation efficiently uses the FPGA resources, while requires no DSP blocks for convolutional operation. Since our architecture employes the on-chip memories (BRAMs) to store weights, inputs, and outputs for all the feature maps, it consumes lower power than the DDR4SDRAM which is widely used for the conventional CNN realization. Also, our implementation achieves higher computation speed than conventional one, since it performs convolutional operation for 512 feature maps at a time. The DDR4SDRAM requires the refresh (pre-charge) operation, while the on-chip memory does not. Since the DDR4SDRAM requires the complex sequences, the memory controller for the DDR4SDRAM tend to be larger area and dissipates higher power than that for the on-chip memory.

VI. Experimental Results

A. Implementation Results

We implemented the binarized CNN for the VGG-16 on the Xilinx Inc. Zynq UltraScale+ MPSoC zcu102 evaluation board, which has the Xilinx Zynq UltraScale+ MPSoC FPGA (ZU9EG, 68,520 Slices, 269,200 FFs, 1,824 18Kb BRAMs, 2,520 DSP48Es). We used the Xilinx Inc. Vivado HLS 2016.4 to generate the HDL source, then used the Xilinx Inc. Vivado 2016.4 with timing constrain 150 MHz. Our implementation used 47,946 Slices, 208,818 FFs, 1367 18Kb BRAMs, and 4 DSP48Es. Also, it satisfied the timing constraint for real-time applications. As for the number of operations for the implemented CNN, sine it performed 3×3 MAC operations for 512 feature maps at 150 MHz, it was 460.80 GOPS (Giga Operations Per Seconds). As for the memory band width, for 512 feature maps, since it reads 3×3 binarized weights, 3×3 binarized inputs, and send a binarized output at a time, it was 139.6 GB per second. We measured the total board power consumption: It was 22 Watt. Since the implemented CNN operated 460.80 GOPS with 47,946 Slices, the area efficiency was 96.1 (GOPS/Slice$\times 10^{-4}$). Also, the performance per power efficiency was 20.94 (GOPS/W).

B. Comparison with other FPGA realizations

Since different implementations used different FPGAs and CNNs, we used the performance per area efficiency (GOPS/Slice) and the performance per power one (GOPS/Power) to do fair comparison. Table III compares various FPGA implementations. From Table III, compared with the recent implementation [35], although the classification accuracy was decreased by 6.5%, our implementation is 2.45 times faster, it is 2.68 times better with respect to the area efficiency, and it is slightly times better with respect for the power efficiency. Thus, our binarized CNN has the highest power-and-area efficiency, which is suitable for the embedded computer vision system.

C. Comparison with other Embedded Platforms

We compared our binarized CNN with other embedded platforms. We used the NVidia Jetson TX1 board which has both the embedded CPU (ARM Cortex-A57) and the embedded GPU (Maxwell GPU). Following the benchmarking [25], the CPU and GPU run the VGG16 using Caffe version 0.14. Also, we measured the total power consumption. Note that, in the experiment, to measure the latency, we set the number of batch size to one.

Table IV compares our FPGA implementation with other platforms. Compared with the ARM Cortex-A57, it was 136.8 times faster, it dissipated 3.1 times much power, and its performance per power efficiency was 44.7 times better. Also, compared with the Maxwell GPU, it was 4.9 times faster, it dissipated 1.3 times much power, and its performance per power efficiency was 3.8 times better. Thus, our method is suitable for the embedded system.

VII. Conclusion

This paper proposed the BN free binarized CNN which treats only binary 2-values (+1/-1) for the inputs and the weights and the integer biases. In this case, the multiplier was replaced into the XNOR gate on the carry-chain of the FPGA. We implemented the large VGG-16 CNN on the Xilinx Inc. Zynq UltraScale+ MPSoC zcu102 evaluation board. Compared with the conventional FPGA realizations, although the classification accuracy was decreased by 6.5%, the performance was 2.45 times faster, the power efficiency was slightly times better, and the area efficiency is 2.68 times better. Compared with the ARM Cortex-A57, it was 136.8 times faster, it dissipated 3.1 times much power, and its performance per power efficiency was 44.7 times better. Also, compared with the Maxwell embedded GPU, it was 4.9 times faster, it dissipated 1.3 times much power, and its performance per power efficiency was 3.8 times better. This is the reason that our binarized CNN stored all the weights, inputs, and output to on-chip BRAMs which is faster and dissipates lower power than off-chip memory, such as a DDR4SDRAM. Thus, our method is suitable for the embedded computer vision system.

VIII. Acknowledgments

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS.

References

[1] Caffe: Deep learning framework, http://caffe.berkeleyvision.org/

[2] S. Chakradhar, M. Sankaradas, V. Jakkula and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *Annual Int'l Symp. on Computer Architecture (ISCA)*, 2010, pp.247-257.

[3] Chainer: A powerful, flexible, and intuitive framework of neural networks, http://chainer.org/

[4] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *Int'l Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[5] The CIFAR-10 data set, http://www.cs.toronto.edu/ kriz/cifar.html

[6] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," *Int'l Conf. on Artificial Neural Networks and Machine Learning (ICANN)*, 2014, pp. 281-290.

[7] CUDA-Convent2: Fast convolutional neural network in C++/CUDA, https://code.google.com/p/cuda-convnet2/

[8] M. Courbariaux, I. Hubara, D. Soudry, R.E.Yaniv, Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *Computer Research Repository (CoRR)*, Mar., 2016, http://arxiv.org/pdf/1602.02830v3.pdf

[9] K. Diederik and B. Jimmy, "Adam: A method for stochastic optimization," *arXiv preprint* arXiv:1412.6980, 2014.

TABLE III. COMPARISON WITH OTHER FPGA REALIZATIONS.

Year	2010 [2]	2014 [15]	2015 [38]	2016 [35]	Proposed
FPGA	Virtex5 SX240T	Zynq XC7Z045	Virtex7 VX485T	Zynq XC7Z045	Zynq UltraScale+ MPSoC ZU9EG
Clock (MHz)	120	150	100	150	150
Memory Bandwidth (GB/s)	—	4.2	12.8	4.2	**139.6**
Quantization Strategy	48bit fixed	16bit fixed	32bit float	16bit fixed	1bit (Binary)
Power (W)	14	8	18.61	9.63	22
Performance (GOPS)	16	23.18	61.62	187.80	**460.80**
Area Efficiency $\times 10^{-4}$) (GOPS/Slice)	4.30	—	8.12	35.8	**96.1**
Power Efficiency (GOPS/W)	1.14	2.90	3.31	19.50	**20.94**

TABLE IV. COMPARISON WITH EMBEDDED PLATFORMS WITH RESPECT TO THE VGG16 FORWARDING (BATCH SIZE IS 1).

Platform	Embedded CPU	Embedded GPU	FPGA
Device	Quad-core ARM Cortex-A57	256-core Maxwell GPU	Zynq UltraScale+ MPSoC
Clock Freq.	1.9 GHz	998 MHz	150 MHz
Memory	16GB eMMC Flash	4GB LPDDR4	32.1 Mb BRAM
Time [msec]	4210.0	156.1	**31.8**
(FPS)	(0.23)	(6.40)	**(31.48)**
Power [W]	7	17	22
Efficiency [fps/W]	0.032	0.376	**1.431**

[10] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *CVPR*, 2015.

[11] A. Dundar, J. Jin, V. Gokhale, B. Martini and E. Culurciello, "Memory access optimized routing scheme for deep networks on a mobile coprocessor," *High Performance Extreme Computing Conf. (HPEC)*, 2014, pp. 1-6.

[12] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," *Int'l Symp. on Circuits and Systems (ISCAS)*, 2010, pp.257-260.

[13] C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," *Int' Conf. on Field-Programmable Logic and Applications (FPL)*, 2009, pp.32-37.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CVPR*, 2014.

[15] V. Gokhale, J. Jin, A. Dundar, B. Martini and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," *CVPR*, 2014, pp.696-701.

[16] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and Vi. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *arXiv prprint arXiv: 1312.6082*, 2013.

[17] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 1, pp.221-231, 2013.

[18] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li, "Large-scale video classification with convolutional neural networks," *CVPR*, pp.1725-1732, 2014.

[19] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, Vol. 86, No. 11, 1998, pp.2278-2324.

[20] Y. Li, Z. Liu, K. Xu, H. Yu and F. Ren, "A 7.663-TOPS 8.2-W Energy-efficient FPGA Accelerator for Binary Convolutional Neural Networks," *Int'l Symp. on Field-Programmable Gate Arrays (ISFPGA)*, 2017, p.290-291.

[21] V. Nair and G.E. Hinton, "Rectified linear units improve restricted Boltzmann machines," *Int'l Conf. on Machine Learning (ICML)*, 2010, pp. 807-814.

[22] H. Nakahara, H. Yonekawa, H. Iwamoto, and M. Motomura, "A Batch Normalization Free Binarized Convolutional Deep Neural Network on an FPGA," *ISFPGA*, 2017, p.290.

[23] H. Nakahara, H. Yonekawa, T. Sasao, H. Iwamoto, and M. Motomura, "A Memory-Based Realization of a Binarized Deep Convolutional Neural Network," *The Int'l Conf. on Field-Programmable Technology (FPT)*, 2016, pp.273-276.

[24] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh and D. Marr, "Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC," *FPT*, pp.1-8, 2016.

[25] https://github.com/charlyng/Embedded-Deep-Learning/tree/master/Benchmark-Performance

[26] M. Peemen, A. A. A. Setio, B. Mesman and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," *Int'l Conf. on Computer Design (ICCD)*, 2013, pp.13-19.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.

[28] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," *Int'l Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, 2009, pp.53-60.

[29] I. Sergey and S. Christian, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[30] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," *CVPR*, pp.1701-1708, 2014.

[31] Theano, http://deeplearning.net/software/theano/

[32] Torch: A scientific computing framework for LUTJIT, http://torch.ch/

[33] A. Toshev and C. Szegedy, "Deeppose: Human pose estimatiion via deep neural networks," *CVPR*, 2014.

[34] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," *ISFPGA*, 2017.

[35] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," *ISFPGA*, 2016, pp. 26-35

[36] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," *ISFPGA*, 2017, pp.65-74.

[37] Xilinx Inc., "7 series FPGAs configurable logic block," *User Guide (UG474)*, Nov. 17, 2014.

[38] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *ISFPGA*, 2015, pp.161-170.

[39] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," *ISFPGA*, 2017, pp.15-24.