Received October 8, 2020, accepted December 7, 2020, date of publication December 31, 2020, date of current version January 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3048375

Binarized Encoder-Decoder Network and Binarized Deconvolution Engine for Semantic Segmentation

HYUNWOO KIM[1], JEONGHOON KIM[2], JUNGWOOK CHOI[3], (Member, IEEE), JUNGKEOL LEE[1], (Student Member, IEEE), AND YONG HO SONG[3,4], (Member, IEEE)

[1]Department of Electronics and Computer Engineering, Hanyang University, Seoul 04763, South Korea
[2]School of Electrical Engineering, Korea University, Seoul 02841, South Korea
[3]Department of Electronic Engineering, Hanyang University, Seoul 04763, South Korea
[4]Samsung Electronics Company Ltd., Hwaseong 18448, South Korea

Corresponding author: Yong Ho Song (yhsong@hanyang.ac.kr)

(Hyunwoo Kim and Jeonghoon Kim contributed equally to this work.)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korea Government (MSIT) under Grant 2020-0-01373, in part by the Artificial Intelligence Graduate School Program, Hanyang University, and in part by the Research and Development Program, Ministry of Trade, Industry and Energy/Korea Evaluation Institute of Industrial Technology (MOTIE/KEIT), "Developing Processor-Memory-Storage Integrated Architecture for Low Power, High Performance Big Data Servers," under Grant 10077609.

ABSTRACT Recently, semantic segmentation based on deep neural network (DNN) has attracted attention as it exhibits high accuracy, and many studies have been conducted on this. However, DNN-based segmentation studies focused mainly on improving accuracy, thus greatly increasing the computational demand and memory footprint of the segmentation network. For this reason, the segmentation network requires a lot of hardware resources and power consumption, and it is difficult to be applied to an environment where they are limited, such as an embedded system. In this paper, we propose a binarized encoder-decoder network (*BEDN*) and a binarized deconvolution engine (*BiDE*) accelerating the network to realize low-power, real-time semantic segmentation. *BiDE* implements a binarized segmentation network with custom hardware, greatly reducing the hardware resource usage and greatly increasing the throughput of network implementation. The deconvolution used for upsampling in a segmentation network includes zero padding. In order to enable deconvolution in a binarized segmentation network that cannot express zero, we introduce *zero-aware binarized deconvolution* which skips padded zero activations and *zero-aware batch normalization embedded binary activation* considering zero-skipped convolution. The *BEDN*, which is a binarized segmentation network proposed to be accelerated on *BiDE*, has acceptable accuracy while greatly reducing the computational and memory demands of the segmentation network through full-binarization and simple structure. *BEDN* has a network size of 0.21 MB, and its maximum memory usage is 1.38 MB. *BiDE* was implemented on Xilinx ZU7EV field-programmable gate array (FPGA) to operate at 187.5 MHz. *BiDE* accelerated the proposed *BEDN* within CamVid11 images of 480×360 size at 25.89 frames per second (FPS) achieving a performance of 1.682 Tera operations per second (TOPS) and 824 Giga operations per second per watt (GOPS/W).

INDEX TERMS Binarized neural network, binarized deconvolution, binarized segmentation network, zero-aware deconvolution, zero-skip deconvolution, neural network accelerator.

I. INTRODUCTION

Recently, the demands for semantic segmentation have grown in various fields such as autonomous driving, surveillance systems, smart factories, and bio-medical image diagnosis. For this reason, many studies have been conducted on semantic segmentation. With the advent of fully convolutional network (FCN) [1], semantic segmentation based on deep neural network (DNN) began to attract attention as it showed higher accuracy and precision compared to segmentation based on classical image processing. After FCN, many studies on segmentation network have been conducted, and segmentation

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Wei.

accuracy and precision have been greatly improved [2]–[5]. However, the segmentation network has a deep and wide structure to enhance accuracy and especially has a network architecture including feature map concatenation or max-unpool layer due to bypass connection, which greatly increases the computation requirement and memory footprint. In an environment where sufficient power is supplied, this segmentation network could be operated through acceleration using an enterprise PC and graphic card. Meanwhile, applications that have to operate in environments with limited computing resources and power budgets, such as autonomous driving and surveillance systems, lack system performance and power, making it difficult to operate segmentation networks.

Therefore, in order to reduce the computational demand and memory footprint of the segmentation network, studies on compact network design (CND) have appeared [6]–[9]. The CND studies improved the network structure to reduce the number of parameters and multiply and accumulation (MAC) operations, enabling the segmentation network to operate on a mobile graphics processing unit (GPU). Nonetheless, since data and operations are basically expressed in and performed on a 32-bit floating-point (or 16-bit fixed-point), there are limits to the increase in performance and reduction in power consumption of the segmentation network implementations. In order to reduce memory and computation overhead due to the large bit-width expression, studies that apply quantization to the weights and activations of segmentation networks have appeared [10]–[12]. The segmentation networks were quantized with various bit-widths under 8-bit, and the computational demands and memory footprints were greatly reduced while minimizing the loss of accuracy due to quantization. Nevertheless, the power consumption was large due to the implementation of the networks on the central processing unit (CPU) or GPU as software, and the performance gain compared to the bit-width reduced by quantization was not large because the bit-lane and memory of computing logic were not efficiently utilized.

In the case of a network with a bit-width of 8-bit or less, the optimal performance and power consumption of network implementation can be derived by designing custom hardware optimized for the network. In the field of image classification, many studies on hardware acceleration of quantized neural networks under 8-bit have been conducted. Among them, the binarized neural network (BNN), which has the highest quantization level, showed the smallest memory footprint and computational demand [13]–[17]. In addition, the BNN custom hardware design has a high compute logic utilization and on-chip memory bandwidth, resulting in the reduction in power consumption and increase in performance compared to the CPU/GPU implementation being very large [18]. In the field of semantic segmentation, there have been studies that accelerated segmentation networks through bit-widths of 8-bit or more with hardware [19]–[24]. These studies accelerated the segmentation network by presenting a hardware architecture for deconvolution. Nonetheless, since they use a large bit-width of 8-bit or more, a large logic is required for operation, and a lot of memory is required to store parameters and feature maps. To the best of our knowledge, studies on the implementation of custom hardware for segmentation networks with a bit-width of less than 8-bit have not been conducted.

Therefore, this paper aims to design a binarized segmentation network and a hardware engine that accelerates it to boost its inference speed and lower the power consumption. To this end, we perform thorough analysis on hardware acceleration of segmentation network and BNN, identify some challenges arising from hardware implementation of binarized segmentation network, and find ways to solve them. First, the segmentation network employs a deconvolution layer unlike the traditional classification network, and deconvolution includes many zero activations due to zero padding. Since BNN cannot express zero, operation and storage of zero activations in deconvolution must be skipped. Second, the number of nonzero activations differs for each convolution sliding window due to zero padding of deconvolution. In the BNN accelerator, batch normalization is embedded in the activation function and processed as thresholding [14], [15]. Due to the variation in the number of nonzero activations in the sliding window, a range mismatch occurs when comparing the threshold and the convolution output. To solve this, it is necessary to perform thresholding in consideration of the number of nonzero activations for each sliding window. Third, the segmentation network has very large feature maps transferred between layers and has a complex structure that maintains feature maps or pool indices while executing multiple layers, such as feature map concatenation or max-unpool, so the memory requirement is very high. Therefore, in order to minimize the memory requirement due to feature maps, a simple segmentation network and hardware architecture minimizing feature map storage are needed.

In this paper, we propose *binarized encoder-decoder network (BEDN)*, a binarized segmentation network with a simple structure, and design *binarized deconvolution engine (BiDE)*, implementing it on field programmable gate array (FPGA) for accelerating *BEDN* to realize the above-described solutions. Our contribution is as follows.

- We design *BEDN* with a simple segmentation network structure focused on considering hardware constraints and minimizing the memory overheads caused by feature maps of network inference. Moreover, we designed *BiDE*, a hardware engine for segmentation inference based on a heterogeneous streaming architecture that consumes all feature maps in on-chip, eliminating off-chip memory accesses of feature maps.
- We propose an encoder-decoder structure with simple unit blocks and organize the network training methodology of BNN with deconvolution layers and a straight-through estimator using hyperbolic tangent for image segmentation neural network.

- We propose the *zero-aware binarized deconvolution* hardware architecture that skips the operation and storage of zero activation and performs binarized deconvolution.
- For the batch normalization and activation function of zero-aware binarized convolution, we propose *zero-aware batch normalization embedded binary activation* that considers the number of nonzero activations in the sliding window and a hardware architecture performing this.
- The accuracy and memory footprint of the proposed *BEDN* are evaluated and compared with other segmentation networks for a representative semantic segmentation dataset. Also, the performance, power consumption, and resource utilization of the proposed *BiDE* are evaluated and compared with the conventional segmentation network accelerator.

This paper is organized as follows. II deals with the background and related works on segmentation networks, BNNs and their hardware implementation. III describes the binarization of segmentation network and *BEDN* structure. IV explains the *zero-aware binarized deconvolution* and *zero-aware batch normalization embedded binary activation* techniques, and describes the *BiDE* architecture. V reports the evaluation and analysis results of *BEDN* and *BiDE*. VI discuss features about proposed *BEDN* and *BiDE*, and VII presents the conclusion of this paper.

II. BACKGROUND

A. PIXEL-WISE SEMANTIC SEGMENTATION NETWORKS

The DNN approaches of pixel-wise semantic segmentation show excellent performance in various benchmark datasets [25]–[27]. In particular, DNN-based semantic segmentation has the advantage of obtaining a segmented image from an input image in an end-to-end manner. This segmentation network mainly has an encoder-decoder structure as shown in Fig. 1. The encoder extracts high-dimensional semantic information through convolution while reducing the spatial dimension by downsampling, and the decoder gradually restores the loss of spatial information due to the spatial dimension reduction through upsampling. An existing classification network model can be adopted as a baseline for the encoder. In general, strided convolution or max-pool is employed for downsampling, and deconvolution or max-unpool is adopted for upsampling. The max-pool stores the indices when pooling in the encoder, max-unpool restores the inputs at the positions indicated by the indices and restores the spatial dimension by filling the remaining pixels with zeros when unpooling in the decoder [4]. Like convolution, deconvolution restores the spatial dimension of a feature map using trainable weight filters and an input feature map [1], [3], [5]. According to the operation method, deconvolution can be classified into *padding free deconvolution* and *zero padding deconvolution* [24]. *padding free deconvolution* performs scalar-matrix products that multiply a single input through elements of a filter matrix, and constructs an output feature map with the resulting windows as illustrated in Fig. 2a. Depending on the stride, windows overlap and the values of this part are summed. *zero padding deconvolution* pads zeros between activations of the input feature map according to the stride and convolves the expanded input feature map and flipped weight filters to obtain the output feature map as shown in Fig. 2b. The decoded feature map is transferred to the last layer and pixel-wise classification is performed. Through this, the segmented image is obtained by classifying the class of each pixel in the image.

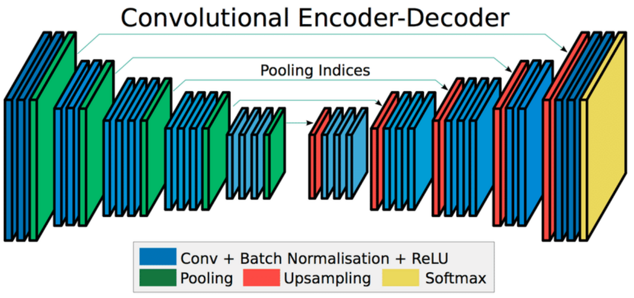
Convolutional Encoder-Decoder
Pooling Indices
Conv + Batch Normalisation + ReLU
Pooling
Upsampling
Softmax

FIGURE 1. Conventional encoder-decoder segmentation network architecture [4].

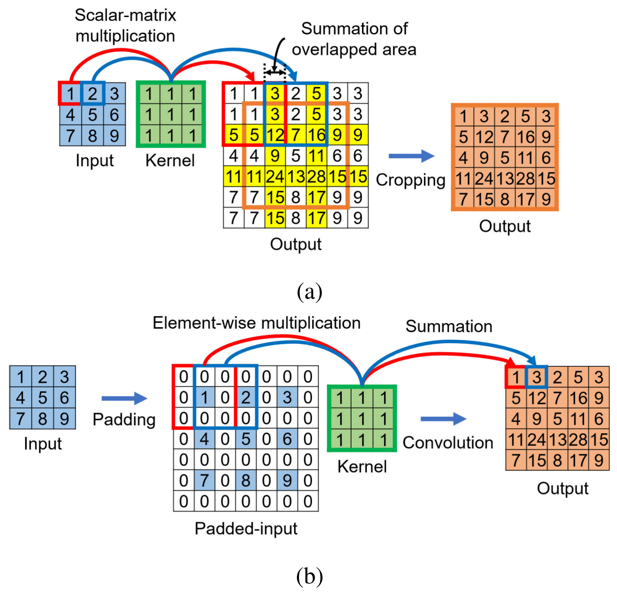

FIGURE 2. Deconvolution methods. (a) Padding free deconvolution, (b) zero padding deconvolution. (Best viewed in color).

B. BINARIZED NEURAL NETWORKS

BNN is a quantized network in which the weights or both of weights and activations are limited to $+1$ and -1, and unlike the traditional network expressed in 32-bit Full-Precision (FP), BNN is expressed in 1-bit. [28]. This BNN is obtained by quantizing activations and weights during the training process of the network. The key to BNN training is to minimize the loss of accuracy of the network by reducing the quantization error, which is the difference between FP weight and 1-bit weight. In the training process, weight binarization is performed by obtaining errors between the ground truths and the results inferred from forward propagation using 1-bit weights and updating the gradients obtained from the errors to the FP weights in backward propagation. This training method increases training time because the weights used in the forward path and the weights utilized in the update path are different, making it difficult to converge the weights. The binarization of activations is achieved by using the sign (signum) function as the activation function. The derivative of the sign function has a problem of the gradients not propagating well because the slopes are zero in all input range except at 0 ($x = 0$) during backward propagation. To solve this problem, straight-through estimator (STE) [28], [29] was proposed. In STE, the derivative is designed so that it has a shape similar to the derivative of the sign function and has slopes in a significant input range, and the problem is solved by replacing the existing derivative with the designed one. At this time, the accuracy and training speed of the network vary according to the shape of the derivative that has replaced the existing one of sign function.

BNN has a loss of accuracy due to loss of information from quantization, but it has been attracting attention as a neural network application solution in environments where power and hardware resources are limited because it greatly reduces the memory footprint and computational demand of neural network inference. BNN binarization can be classified into partial-binarization, which binarizes only weights, and full-binarization, which binarizes both weights and activations. Partial-binarization [30], [31] has the advantage of greatly reducing the weight size so that all parameters can be stored in the on-chip memory, and the loss of accuracy is small because the activations are maintained in FP. Full-binarization [28], [32], [33] has the advantage of significantly reducing the memory footprint by reducing the size of feature maps transferred between layers while taking more loss of accuracy by binarizing activation.

C. BINARIZED NEURAL NETWORK OPERATION OPTIMIZATIONS

BNN can reduce the number of computations and hardware resources used for computations by optimizing neural network operations because data is limited to $+1$ and -1. Operations optimized in BNN include MAC operation, batch normalization operation, and activation function. MAC operation is the main operation of convolution, which multiplies input activation and weight, accumulating it in units of the sliding window. There are only 4 multiplication operations possible with data consisting of $+1$ and -1, as shown in Table. 1. Here, if -1 is expressed as 0, data can be expressed in 1-bit, and the multiplication operation is the same as XNOR. Therefore, the multiplication of BNN can be implemented as an XNOR gate instead of a multiplier with a large logic size [28], [32].

TABLE 1. Relationship between +1, −1 multiplication and XNOR.

Multiplication			XNOR		
Activation	Weight	Output	In A	In B	Output
-1	-1	1	0	0	1
-1	1	-1	0	1	0
1	-1	-1	1	0	0
1	1	1	1	1	1

Accumulation is the addition of all the multiplication results in the sliding window, and the multiplication results are only $+1$ or -1, so accumulation can be calculated with *the number of ones* $-$ *the number of minus ones* in the sliding window. Since the number of -1 is *fan-in of sliding window* $-$ *the number of ones*, accumulation can be obtained by simply counting the population (popcount), which is the number of ones in the sliding window. Here, fan-in refers to the number of elements of the sliding window. Therefore, the accumulation of BNN can be implemented with popcount logic. Since BNN multiplication is carried out by XNOR, a multiplication result of -1 is output as 0. Popcount can be implemented in signed popcount [14], [16], [34] which accumulates by changing 0 into -1 and unsigned popcount [15], [35] which adds 0 as it is.

Batch normalization is performed through complex mathematical calculations such as (1) using the average μ and variance σ of the input batch, and the learned parameters γ and β.

$$Y_{Norm} = \gamma \frac{(Y - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{1}$$

Y refers to a convolution output calculated through MAC operation in a sliding window unit, and Y_{Norm} refers to a normalized convolution output. If the batch normalization formula is summarized for Y, it becomes (2).

$$\begin{aligned} Y_{Norm} &= \frac{\gamma}{\sqrt{\sigma_2 + \epsilon}}\{Y - (\mu - \frac{\sqrt{\sigma^2 + \epsilon}}{\gamma}\beta)\} \\ &= \lambda\{Y - (\mu - \frac{\beta}{\lambda})\} \end{aligned} \tag{2}$$

For simplicity of expression, in this paper, the scale term $\gamma/\sqrt{\sigma_2 + \epsilon}$ is expressed as λ. BNN performs binarization and activation of the convolution output through the sign activation function.

If the network architecture is an architecture in which the activation function follows batch normalization, the sign activation function is equal to (3) and activates when the normalized convolution output (Y_{Norm}) is greater than or equal to 0.

$$Activation = \begin{cases} 1, & Y_{Norm} \geq 0 \\ 0, & otherwise \end{cases} \tag{3}$$

Since only the sign of Y_{Nrom} is needed in the sign activation function, the λ scale of (2) can be ignored. Eventually, if λ is positive, the sign activation function is to activate if $Y - (\mu - \beta/\lambda) \geq 0$, so it can be expressed as (4).

$$Activation = \begin{cases} 1, & Y \geq \mu - \frac{\beta}{\lambda} \\ 0, & otherwise \end{cases} \tag{4}$$

If λ is negative, that is, if γ is negative, the inequality sign of (4) is reversed. By using $\mu - \beta/\lambda$ of (4) as a threshold and activating only Y that exceeds the threshold, batch normalization and sign activation can be performed at once [14], [15]. We call this as *batch normalization embedded binary activation (BNEBA)*.

μ, σ, γ, and β are parameters determined during training. Therefore, since the thresholds can be calculated in advance off-line, the complex operations of batch normalization do not need to be carried out at run-time. When unsigned popcounts are utilized during calculating convolution outputs, it is necessary to make the ranges of convolution outputs and thresholds equal. In the case of using unsigned popcount, the range of the convolution outputs is [0, +fan-in], but the thresholds use signed accumulation during training, so the range is [−fan-in, +fan-in]. Therefore, the thresholds are shifted and scaled like (5) to match the convolution outputs and the thresholds ranges [15].

$$th_{new} = (th_{old} + fan\text{-}in)/2 \tag{5}$$

th_{old} is the threshold precompute during training, and th_{new} is the threshold in which the range is matched to the convolution output.

D. RELATED WORKS

Early segmentation networks were developed in the direction of increasing accuracy, and this resulted in complex structures with a large number of parameters and operations [1]–[5]. These networks are based on the encoder-decoder structure, and the multi-scale feature maps generated by the encoder are concatenated in the decoder [1], [3], [5] or either max-unpool is used in the decoder [2], [4]. Such a complex structure requires the storage of feature maps or pool indices, resulting in a very large memory footprint. Therefore, these networks require high-end graphics card acceleration such as NVIDIA Titan [3], [4] or Tesla K40c [1] even for inference. In the case of FCN [1], the PASCAL Visual Object Classes (PASCAL VOC) 2011, 2012 test sets are inferred with the latency of < 175 *ms* on NVIDIA Tesla K40c.

In order to reduce the computational demand and memory footprint of a segmentation network, studies have emerged to reduce bit-width by quantizing the activations and weights of the network. Fixed point U-Net [10] proposed U-Net quantized with various bit-widths from 16-bit to partial binary. This study reduced the memory requirement by 8 times compared to FP with a 4-bit weight, 6-bit activation network, and showed only dice score losses of 2.21%, 0.57% and 2.09% for the spinal cord gray matter segmentation (GM), electron microscopic (EM), and public national institute of health (NIH) datasets, respectively. Quantized FCN [11] quantized FCN to 7, 5, and 3-bit. This study reduced memory usage by 4.6× and improved dice score by 1% compared to FP network [36] with a 7-bit network for the medical image computing and computer assisted interventions (MICCAI) Gland dataset. The binarized FCN [12] partially binarized the FCN, and showed 7% and 4.7% mean intersection over union (mIoU) loss for PASCAL VOC 2012 and Cityscapes datasets, respectively, and inferred 256 × 160 images on Tegra K1 CPU at a rate of 0.81 frames per second (FPS). However, these studies still have large memory footprints by using complex network structures such as FCN and U-Net. Furthermore, because the quantized neural network was implemented as software on the CPU or GPU, the bit-lane utilization of the compute logic was poor, with limited performance gain.

Therefore, in order to accelerate the quantized segmentation network and reduce the power consumed by it, many studies have appeared in accelerating by custom hardware. The main difference between the traditional classification networks and the segmentation networks is the deconvolution layer, so these studies focused on the acceleration of deconvolution, presenting hardware architectures for *padding free deconvolution* or *zero padding deconvolution*. Liu *et al.* [20] and FCN-engine [23] accelerated the *padding free deconvolution* of an 8-bit network with FPGA and application specific integrated circuit (ASIC), respectively. Liu *et al.* inferred the 256 × 256 images at 1.79 FPS/W. PAI-FCNN [22], DT-CNN [21], and RED [24] accelerated *zero padding deconvolution* by hardware, and took approaches to skip the operation of zero activation. PAI-FCNN accelerated a 3-bit weight, an 8-bit activation network with FPGA and inferred 300×300 images at 10.1 frames per second per watt (FPS/W). DT-CNN designed an ASIC that accelerates an 8-bit network and inferred 288 × 288 images at 349 FPS/W. The high FPS/W of DT-CNN was possible because of the very low number of operations in network and power consumption. RED designed a memristor that accelerates the Resistance Random Access Memory-based (ReRAM-based) FP network. Deconvolution is an operation that is also adopted in generative adversarial networks (GAN), and there have been many studies on deconvolution accelerators for GANs. Wang *et al.* [37], Song *et al.* [38], GANAX [39], LerGAN [40] accelerated the 16-bit network, and GNA [41] implemented deconvolution accelerator that supports flexible bit-width of 8-bit and 16-bit.

We have noticed several factors that hinder performance and increase power consumption in conventional quantized segmentation networks and hardware accelerations. Traditional quantized segmentation networks have large memory requirements due to feature map concatenation or max-unpool. According to our calculation, the maximum memory usage of 480 × 360 image inference of CamVid dataset [42] is 416 mega bytes (MB) for FCN, 45 MB for SegNet, and 28 MB for DeepLabV3+. The size of the feature maps can be greatly reduced by performing binarization, but the segmentation networks have been studied only down to partial binarization, and there has been no research on full binarization. Researches on hardware acceleration have only been conducted down to 8-bit using large logic for computation. In addition, traditional segmentation network accelerator studies process layers sequentially, and because feature maps are large, off-chip memory accesses occur due to feature maps transferred between layers. The resulting power consumption and latency are expected to be large, but existing studies have not considered it [21]–[23], [37], [39], [41]. Liu *et al.* [20] and Song *et al.* [38] store all feature maps in on-chip memory. This was possible because the former significantly reduced the image size and network channel size, and the latter was GAN, the input image and network channel size being small. In general, a segmentation network has large-sized inputs and large-sized channels, so the size of feature maps is large. Our proposed *BEDN* reduces the memory footprint through a simple structure and greatly reduces the parameter size, feature map size, and computational demand through full binarization. Moreover, *BiDE*, which accelerates *BEDN*, is designed based on a heterogeneous streaming architecture, and processes all feature maps transferred between layers in on-chip, showing high performance and low power consumption through binarized convolution and deconvolution operations. Table. 2 summarizes the advantages and disadvantages of previous implementations and ours.

TABLE 2. Pros and Cons of various segmentation network implementations.

Implementations	Pros	Cons
SW on GPU (Under 8-bit) [10]–[12]	High flexibility High programmability	Low power efficiency Low HW utilization Large memory footprint
HW accelerators (8-bit) [20]–[24], [37]–[41]	High power efficiency High HW utilization	Low flexibility Low programmability Large memory footprint
Ours (1-bit)	Highest power efficiency Highest HW utilization Small memory footprint	Low flexibility Low programmability Low accuracy

III. BINARIZED ENCODER-DECODER NETWORK

This chapter describes the architecture of the proposed *BEDN* and its binary-quantization that reduces the computational demand and memory footprint of the network for *BiDE*.

A. NETWORK ARCHITECTURE

BEDN architecture is illustrated in Fig. 3. The architecture of *BEDN* is designed in consideration of hardware constraints. There are two representative methods of configuring a layer of a fully binarized neural network. The first is a method applied to *BEDN*, which was first proposed by Hubara *et al.* [28], consisting of a unit block of Fig. 3a. This unit block performs sign activation at the end so that the 1-bit binarized activations are transferred between layers. The second method was proposed in Xnor-Net [32] and Bi-real net [33] to improve accuracy, and its architecture is depicted in Fig. 3b. This architecture binarizes the activations through the sign activation function just before the convolution operations, so 1-bit activations are only used during the convolution operations. After convolution, activations are restored to FP by multiplying the convolution outputs by the scale parameters and performing batch normalization, so FP activations (FP feature map) are transferred between layers. Therefore, there is an advantage of designing a network model with higher accuracy using this FP feature map. Even so, in the case of a segmentation network, the size of feature maps is much larger than that of the classification network, so designing a segmentation network based on the unit block in Fig. 3b increases the memory footprint a lot. In this regard, as in Yang *et al.* [43], the memory footprint can be reduced by grouping two unit blocks and streaming the FP feature map without storing it in memory. However, since the FP feature map cannot be utilized, this method has the same model as the network model using the unit block of Fig. 3a. Therefore, *BEDN* is designed based on the unit block of Fig. 3a to reduce the memory footprint.

BEDN has an encoder-decoder structure in which the unit block of Fig. 3a is repeated as in Fig. 3c. The encoder extracts features of the input image using binarized convolution, and the decoder expands the features using binarized deconvolution, creating a segmented image of the same size as that of the input image. *BEDN* has a simple and repetitive architecture like SegNet or U-Net but does not employ bypass connections such as feature map concatenation of U-Net or max-unpool of SegNet to reduce the memory footprint. The *BEDN* model is shown in Table. 3. The basic model consists of a total of 11 convolution layers. Layer 1–6 compose encoder and extract features of the input image while reducing the spatial dimension by performing strided convolution in layers 3 and 5. In the case of using the max-pool for spatial dimension reduction in a network with 1-bit activations, since activations have only one of two values -1 and $+1$, many values in the pooling window are maximized, greatly reducing the effect of the max-pool. Therefore, *BEDN* utilizes strided convolution instead of the max-pool. Layer 6–11 constitute decoder and restore the spatial dimension by expanding features through deconvolution in layers 7 and 9.

TABLE 3. *BEDN* model for CamVid11 dataset segmentation.

Layer	K	ICH	OCH	IF	St	L-GOPs	P-GOPs
1 (C)	3	3	64	480×360	1	0.30	0.30
2 (C)	3	64	64	480×360	1	6.37	6.37
3 (C)	3	64	128	480×360	2	3.19	3.19
4 (C)	3	128	128	240×180	1	6.37	6.37
5 (C)	3	128	256	240×180	2	3.19	3.19
6 (C)	3	256	256	120×90	1	6.37	6.37
7 (D)	3	256	128	120×90	1 (2^a)	12.74	3.19
8 (C)	3	128	128	240×180	1	6.37	6.37
9 (D)	3	128	64	240×180	1 (2^a)	12.74	3.19
10 (C)	3	64	64	480×360	1	6.37	6.37
11 (C)	3	64	11	480×360	1	1.10	1.10

C = convolution layer, D = deconvolution layer, K = kernel size, ICH = input channel size, OCH = output channel size, IF = input feature map size, St = strides, L- & P-GOPS = logical and physical number of Giga operations. The kernel is always symmetric.

^{a} The stride of the deconvolution layer used conventionally. This represents the information of zero padding between input activations.

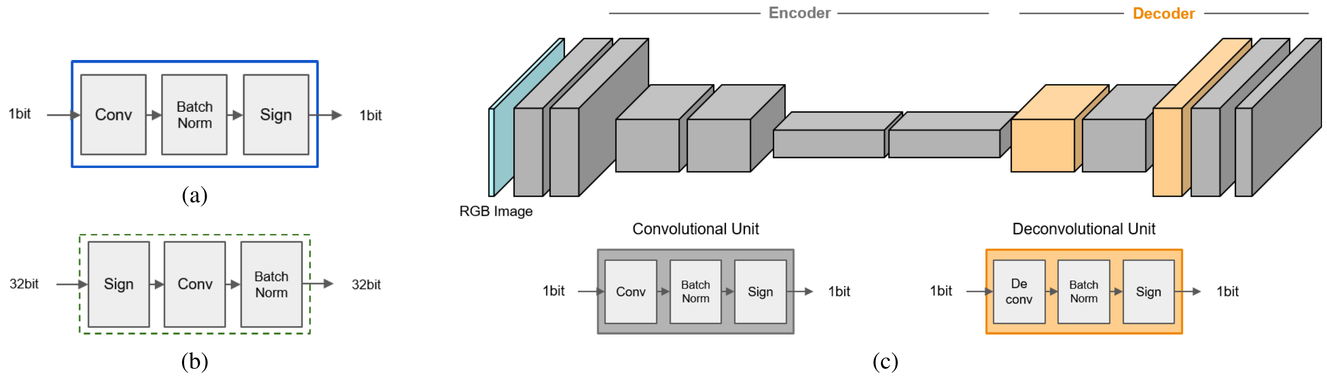

FIGURE 3. ***BEDN*** **architecture. (a) 1-bit Unit Block, (b) 32-bit Unit Block, (c)** ***BEDN*** **model.**

The convolutional unit of the last layer does not have an activation function to obtain a segmented image consisting of scores of each class, thus the normalized convolution results being output as scores. In this *BEDN* architecture, the model can be easily changed by repeating and removing unit blocks, and the designer can intuitively modify the model to trade-off the accuracy and the number of operations (OPs). The number of OPs for each layer in Table. 3 is calculated using (6).

$$OPs = (ICH \times OCH \times IW \times IH \times K^2)/St^2 \qquad (6)$$

The number of physical giga operations (GOPs) is calculated by considering the input feature maps excluding padded zeros, and the number of logical GOPs is calculated by considering the input feature maps including padded zeros in the deconvolution layer.

B. NETWORK TRAINING

BEDN is a fully binarized neural network, and activations and weights are binarized by applying network quantization techniques in the training process. Training of *BEDN* follows the binarization methodology proposed in Hubara *et al.* [28]. The binarization method used for the deconvolution layer is the same as the convolution layer. Consequently, training of *BEDN* is performed by forward propagation with binarized weights to obtain errors and updating FP weights in backward propagation while propagating gradients derived from the errors. *BEDN* utilizes the sign function as an binary activation function for forward propagation, and uses the derivative of the hyperbolic tangent function modified by the STE method for backward propagation.

The activation functions and its derivatives used for training BNN are shown in Fig. 4. Sign function and its derivative are shown in the Fig. 4a, which is zero in all range except at zero ($x = 0$), and gradients are not transmitted in this range. Fig. 4b is the method proposed in Hubara *et al.* [28], so that gradients can be transmitted in the range of $[+1, -1]$ using the derivative of hard hyperbolic tangent (tanh). The proposed method shown in Fig. 4c prevents disconnection that appears at the boundary of $+1$ and -1 in Fig. 4b and helps gradients to be transmitted continuously in backward

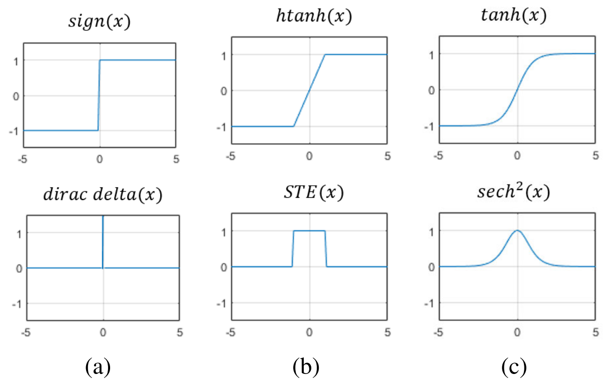

FIGURE 4. **Forward and backward approximations of sign activation function. (a) actual forward and backward functions of sign activation. (b) STE approximation [28]. (c) our modified approximation.**

propagation, which boosts the training speed and improves the accuracy [29]. The derivative that we employed can be expressed as (7), where x is an input and the scale α is the hyper-parameter with a default value of 1. The scale α affects the training convergence speed and final accuracy.

$$\frac{d\tanh(x)}{dx} = \alpha \times \sec^2(x), \quad \alpha > 0 \qquad (7)$$

When network training is completed, trained batch normalization parameters are embedded into the thresholds to apply the *BNEBA* which is described above in II-C. The embedding of the batch normalization parameters is performed by calculating the threshold of (4). In the case of networks using multi-bit weights, batch normalization parameters can be folded to the weights of the convolution layer, as in Jacob *et al.* [44]. Meanwhile, in *BEDN*, since the weights are binary, other information cannot be folded into the weights, so *BNEBA* is used.

IV. BINARIZED DECONVOLUTION ENGINE

We design *BiDE* in order to accelerate *BEDN*. *BiDE* performs deconvolution and *BNEBA* considering padded zeros for binarized deconvolution. This chapter describes the baseline architecture of *BiDE*, the binarized deconvolution engine architecture, and the hardware architecture for other operations in detail.

A. BASELINE ARCHITECTURE

BiDE is designed based on FINN [15] architecture. FINN is a heterogeneous streaming architecture as illustrated in Fig. 5a. Each layer has a dedicated compute array, and each array is shaped by considering the number of operations and latency of the layer. Therefore, the hardware structure is specific to the network architecture, but since it is an FPGA implementation, it is possible to cope with network architecture changes. As depicted in Fig. 5b, compute arrays do not wait until the previous layer's operations are completely finished, and as soon as the previous layer's outputs are produced, it starts the operations immediately by using it as inputs. Data transfers between compute arrays are streamed through first in first outs (FIFOs). In this structure, since the feature maps, which are the layer outputs, are immediately consumed in the next layers, memories for storing the feature maps other than shallow depth FIFOs and small buffers are not required.

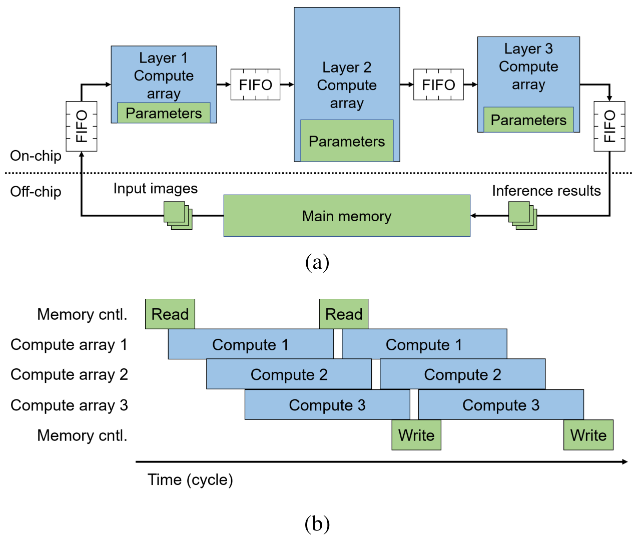

FIGURE 5. Overall architecture and scheduling. (a) heterogeneous streaming architecture, (b) pipelined layer scheduling.

On the other hand, as explained in II-D, conventional accelerators execute one layer completely and then execute the next layer. Therefore, it is necessary to store the entire feature maps transferred between layers, and because the size of the feature maps is large, a lot of off-chip memory accesses are required. In order to reduce such off-chip memory accesses and reuse feature maps in on-chip memory as much as possible, there has been a study that first processes operations in pyramid units composed of multi-layer tiles as depicted in Fig. 6a [45]. If the pyramid is constructed so that the outputs (features) of each tile can be sufficiently stored in the on-chip memory, the features created in each layer can be immediately consumed by the next layer and be discarded. Venkataramani *et al.* [46] also attempted to eliminate off-chip memory accesses by scheduling the layer executions to the

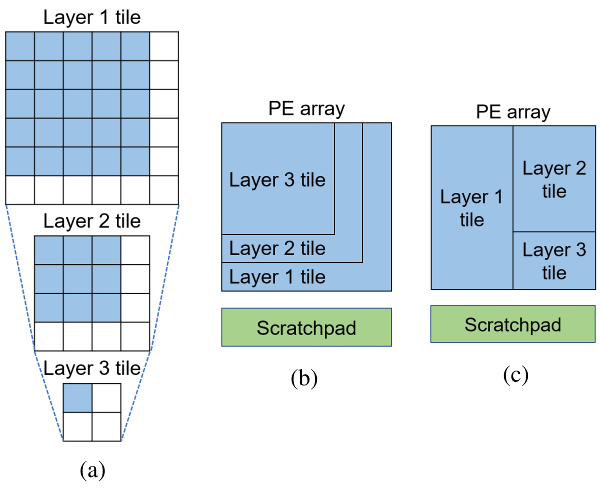

FIGURE 6. Multi-layer execution. (a) computation pyramid across multi-layer, (b) temporally sequenced layer execution on a systolic array, (b) spatially pipelined layer execution on a systolic array.

systolic array to reuse the feature maps as much as possible in the next layers. If the layer tiles of Fig. 6a are sequentially scheduled in the systolic array, it is the same as Fig. 6b. After computing the layer 1 tile, the layer 2 tile is computed by using the outputs of layer 1 tile as the inputs, and the outputs of the layer 1 tile are discarded. In this method, the features can be reused in the processing element (PE) array registers by allocating the next layer tile to the PEs in which the outputs of the previous layer tile are stored, having no need to store the features in scratchpad memory. However, as in Fig. 6a, the workloads of layer tiles are different, so the utilization may not be good because the workloads are not fit to the PE array. To avoid this, multi-layer workloads can be allocated spatially to fit the PE array, and features can be reused in on-chip scratchpad memory by simultaneously processing them in a pipeline manner as depicted in Fig. 6c. This method stores all the computation results of each layer tile in the scratchpad memory. Although it is much smaller than the off-chip memory access, the latency of accessing the scratchpad memory is nontrivial. Assuming the size of layer 1, 2, and 3 tiles to be $6 \times 6 \times 16$, $4 \times 4 \times 24$, $2 \times 2 \times 32$, respectively, and the scratchpad memory to be a single port memory, 960 input reads and 512 output writes occur during layer 1 and 2 tile execution, and a total of 1472 cycles are required. (read: $6 \times 6 \times 16 + 4 \times 4 \times 24$, write: $2 \times 2 \times 32 + 4 \times 4 \times 24$) This assumes that all weights are reused in PE, so scratchpad memory accesses occur only when reading inputs/outputs of each layer tile. Thus, the number of cycles is calculated by considering only the reads and writes of activations (features). Also, since this method operates in units of layer tiles, the next layer tile operation can be started only after the previous layer tile operation is completed. Besides, the time for reading the inputs and filling the PE array before starting the execution of the PE array and the time for writing the outputs after the execution is completed are required other than the execution time. Therefore, assuming that multiply takes 3 cycles, addition takes 1 cycle, 3 × 3 kernel is used, and 32 and 16 PEs with 8 single instruction multiple data (SIMD) lanes are allocated to layer 1 and 2 tiles respectively, computation takes 1728 cycles ($= 3 \times 3 \times 16 \times 24 \times 4 \times 4 \times (3+1)/(32 \times 8) + 3 \times 3 \times 24 \times 32 \times 2 \times 2 \times (3+1)/(16 \times 8)$), and a total of 3200 cycles (compute: 1728, read: 512, write: 960) are required to process layer 1 and 2 tiles.

The heterogeneous streaming architecture has compute arrays dedicated to each layer, so the array is fully utilized. Layer outputs between compute arrays are not stored in memory, and features as many as SIMD lanes are directly streamed in bulk through wide FIFOs. Also, as soon as the number of features equal to the number of SIMD lanes is created, the computations of the next layer can be started. Therefore, in the case of a heterogeneous streaming architecture that has the same number as the number of SIMD lanes and PEs used in the assumption of the Fig. 6c architecture in the above paragraph, the number of computation cycles is the same, but the number of cycles to read/write features proportionally decreases as much as they are transferred in bulk. In other words, if the assumption applied to the cycle calculation of the Fig. 6c architecture in the previous paragraph is applied to the heterogeneous streaming architecture as it is, it takes 184 cycles ($= 1472/8$) to read/write and a total of 1912 cycles (compute: 1728, read: 64, write: 120) to process layer 1 and 2 tiles. Therefore, for the purpose of this paper, which is low power and real-time segmentation inference, a heterogeneous streaming architecture with low latency and low memory overhead due to the feature maps and high utilization of computing resources is adopted as the baseline architecture.

The compute array of baseline architecture illustrated in Fig. 5a is mainly composed of *Matrix-Vector Thresholding Unit (MVTU)* and *Sliding Window Unit (SWU)*. *MVTU* is a unit in charge of binarized convolution and is a computational core. *MVTU* performs the XNOR-popcount operations of the weight matrices and the input activation vector, accumulates the generated partial sums, and performs thresholding on accumulations. The XNOR-popcount operation corresponds to the MAC operation, and the thresholding corresponds to the batch normalization and activation function of the conventional convolutional neural network (CNN). *MVTU* employs unsigned popcount, so scaling and shifting of (5) are applied to the thresholds. The structure of *MVTU* is depicted in Fig. 7. It consists of FIFOs for input/output activation stream, input/output buffer, and P processing elements (PE). PEs create different output channels in parallel using different weight filters for the same input feature map. The input activation stream is broadcasted to all PEs. Each PE has S SIMD lanes, and performs XNOR-popcounts on S input channels in parallel to obtain partial sums, and then, accumulates the partial sums. When the XNOR-popcount operations for one sliding window are completed, the accumulated partial sum, that is, the convolution output, is normalized and activated through thresholding and stored in the output buffer. Weight memories and threshold memories are distributed to each PE. The weight memories and threshold memories of each

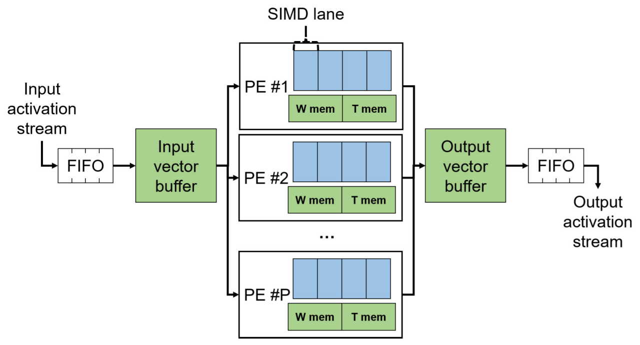

FIGURE 7. Matrix-vector thresholding unit.

MVTU store all the weights and thresholds required for the operations of the corresponding layer. Since every layer has a dedicated *MVTU*, all network parameters (weights and thresholds) are stored in on-chip memory. Therefore, off-chip memory accesses occur only when reading input images, writing inference results, and bringing weights and thresholds once to the on-chip memory initially. In the case of the *BEDN* model shown in III-A, the size of weights and thresholds of the entire layer is only 213 kB. All other reads/writes are resolved in on-chip memory.

For parallel execution in the SIMD lanes of *MVTU*, feature maps and weight filters are streamed and stored in a channel interleaving manner as depicted in Fig. 8a. The left side of Fig. 8a is an example of channel interleaving of the feature map when the 3 × 3 × 4 feature map is simultaneously processed by two SIMD lanes. Each element indicated by N_i represents the N-th element of the i channel. In this example, since the number S of SIMD lanes is two, two input channels, that is, 2-bit, are processed as one word and interleaved in units of two channels. The right side of Fig. 8a shows an example of channel interleaving of the weight filter when four 2 × 2 × 4 weight filters are processed in parallel with two SIMD lanes ($S = 2$) on two PEs. Since the number of PEs P is two, four filters are stored in two weight memories, and like the input feature map, they are interleaved and stored in units of two channels.

SWU creates an input activation stream used in *MVTU* from the output activation stream of the previous layer. *MVTU* consumes all partial sums and generates output by calculating in units of the sliding window like Eyeriss [47]'s output stationary method. Therefore, *SWU* creates an input stream in units of the sliding window in the order of accesses from *MVTU*. In this process, the overlapping parts of the windows are redundantly streamed. Due to this, it seems that bandwidth is consumed redundantly, but even when input activation is reused in on-chip memory or register, one read cycle is required to utilize the redundant activation, so the latency is the same in both cases. The difference between the two cases is that in case of reuse, the activation is written once to the memory or register and then read several times, that is, reused, and in the case of streaming, the activation is written several times and also read several times. However, since

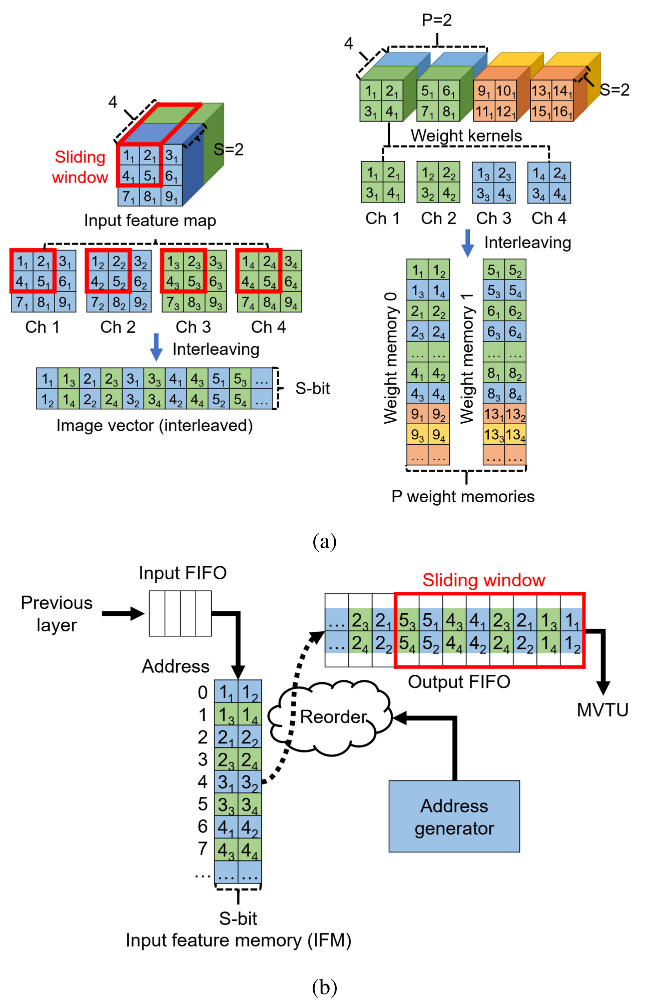

FIGURE 8. **Streaming method. (a) channel interleaving, (b) *SWU* operation. (Best viewed in color).**

FIFO has separate read/write ports, it is possible to read/write at the same time, and writes caused by stream generation operate in a pipelined manner with MVTU, so that the latency can be hidden.

Fig. 8b illustrates the operation of *SWU*. In the input feature map (IFM) memory, channel interleaved output feature maps streamed from the previous layer are stored in a raster scan order. The address generator creates IFM memory addresses while sliding the window. *SWU* reads input activations from IFM memory with the created addresses and streams the reordered input activation vector to *MVTU*. IFM memory needs to hold the input feature map rows as much as the sliding window height for reordering the input feature map, so it requires $(\textit{kernel height} + 1) \times \textit{input feature map width} \times$ *the number of input channels* bits including one IFM row for prefetching.

B. ZERO-AWARE BINARIZED DECONVOLUTION

As described above in II-A, deconvolution can be classified into *padding free deconvolution* and *zero padding deconvolution*. In *padding free deconvolution*, the overlapping portions of the windows must be added, so the multiplication results of multiple windows must be held for a long time. Since this requires a lot of buffers in the streaming architecture, we adopted *zero padding deconvolution*. *zero padding deconvolution* is performed through upsampling the input feature maps by padding zeros, and then convolve the expanded input feature maps and flipped weight filters. Since this is a convolution operation, it seems that it can be calculated with *MVTU* and *SWU* of the baseline architecture, but this is impossible. This is because BNN is represented by only two numbers $+1$ and -1, so zero cannot be expressed. Since the zero used in *MVTU* and *SWU* refers to -1, *MVTU* cannot perform deconvolution including the actual zero. Therefore, in order to enable deconvolution operation in BNN, we propose *zero-aware binarized deconvolution* that skips operations and storage of zero activations and design streaming hardware that can perform this.

To skip deconvolution's operations and storage of zero activations, we need to know the positions of padded zero activations in the input feature maps. The zero padding of deconvolution uniformly inserts zeros between pixels of the input feature map according to the downsampling rate (stride) of the layer corresponding to the deconvolution layer in the encoder. That is, the locations of padded zeros are determined according to the network architecture, and the locations of zero activations can be known in advance without performing zero detection. Since *BiDE* is dedicated to the network architecture, each compute array performing deconvolution can be designed to separately skip zero activations.

The compute array that performs the operations of *zero-aware binarized deconvolution* is composed of *Zero-Aware Sliding Window Unit (ZASWU)* and *Zero-Aware Matrix-Vector Thresholding Unit (ZAMVTU)*. *ZASWU* skips zero activations in the zero padded feature maps and creates a nonzero activation input stream to be used in *ZAMVTU*. Zero padding in deconvolution is performed to restore the dimension as much as it is reduced in the downsampling layer. Fig. 9 shows downsampling due to strided convolution, and Fig. 10 shows sliding windows of the deconvolution layer corresponding to Fig. 9. The input feature map is the

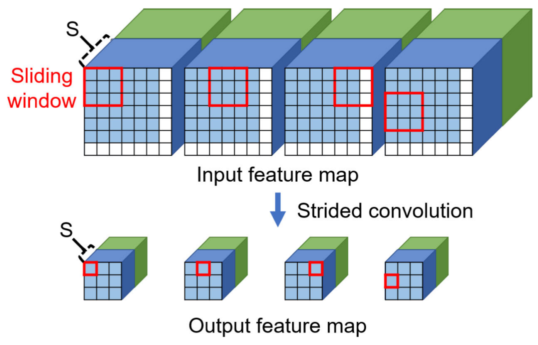

FIGURE 9. **Downsampling using strided convolution with 2 strides, 3 × 3 kernel, and bottom, right zero edge padding.**

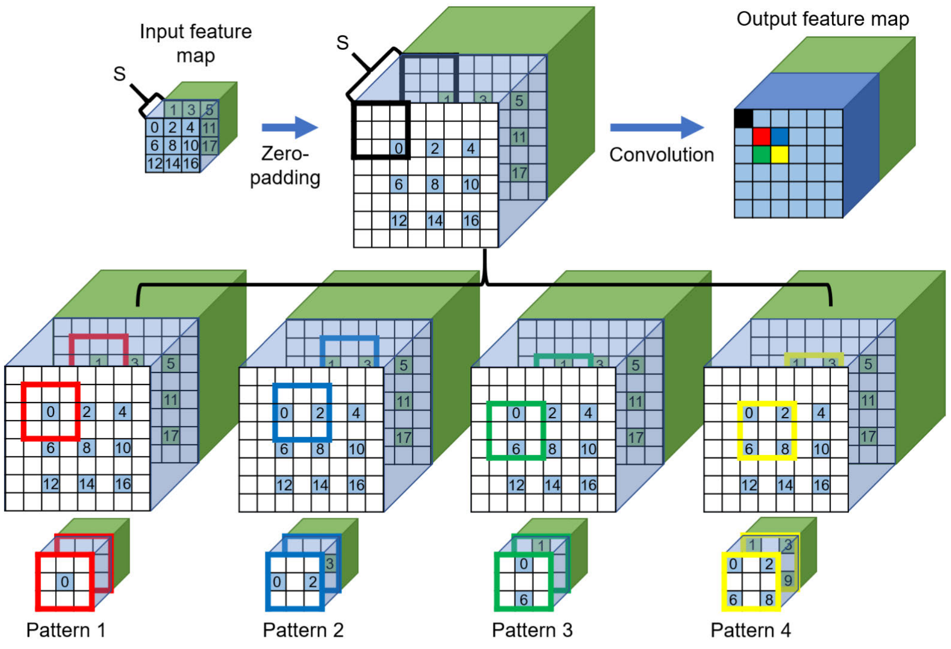

FIGURE 10. Sliding window patterns of deconvolution. (Best viewed in color).

output feature map of the previous layer, and the input feature map is upsampled by padding zeros for deconvolution. The convolution is carried out by sliding the window on the zero padded input feature map. In the example network architecture, the sliding window is repeated in four patterns. These four patterns can be associated with the spatial coordinates of each sliding window output in the output feature map. When the spatial coordinates of the sliding window output are expressed as (row, column), the sliding window is pattern 1 for (odd, odd), pattern 2 for (odd, even), pattern 3 for (even, odd), and pattern 4 for (even, even). In the case of a sliding window including edge padding, the padded edges are separately skipped (black pixel and black box).

Fig. 11 shows the input stream generation of *ZASWU*. The output feature map of the previous layer is streamed to *ZASWU* through the input FIFO. The output feature map of the previous layer, that is, the input feature map of the current layer interleaved by S channels in raster scan order, is popped out from the input FIFO and stored sequentially in the IFM memory. IFM memory is a line buffer with S-bit width, and addresses are assigned using S-bit as words. *ZASWU* reads the nonzero activations of each sliding window pattern from the IFM memory while sliding the window, creates a nonzero activation stream, and stores it in the output FIFO. To this end, *ZASWU* creates IFM memory addresses of nonzero activations based on the output spatial coordinates and the patterns of the sliding windows. In the example of Fig. 10 and Fig. 11, when the output spatial coordinates are (1,1) (red pixel), the sliding window pattern is 1, so the IFM memory addresses are created in the order of 0, 1. When the window slides, the output spatial coordinates are (1,2)

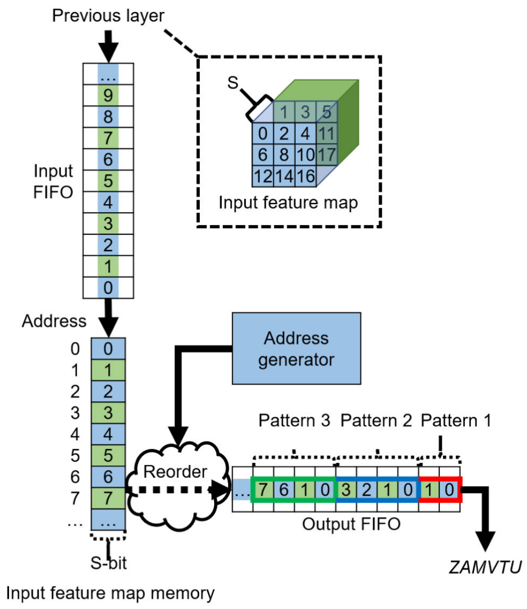

FIGURE 11. Nonzero input stream generation of *ZASWU*.

(blue pixel), and the sliding window pattern is 2, so the IFM memory addresses are generated in the order of 0, 1, 2, 3. If the output spatial coordinates are (0,0) (black pixel), the pattern is 4, but since zero edge padding is included, IFM memory addresses 0, 1, 2, 3, 6, 7 are skipped and only 8 and 9 are generated. *SWU* creates IFM memory addresses while sliding the window in raster scan order from output spatial coordinates (0,0), reads nonzero activations from IFM memory using the created addresses, and streams them to *ZAMVTU*. Since *ZASWU* creates an input stream from feature maps where zeros are not padded, IFM memory only needs spaces as much as $(\textit{filter height} - \textit{stride} + 2) \times \textit{input feature map width} \times \textit{the number of input channels}$ bits for input reordering. It includes IFM rows of $\textit{filter height} - \textit{stride} + 1$ for the sliding window, and one IFM row for prefetching. Therefore, *ZASWU* has the effect of reducing the size of the IFM memory by skipping the storage of zero activations.

ZAMVTU performs zero-skipped convolution by using the input activation stream generated in *ZASWU*. Since *ZASWU* only streams nonzero activations in the sliding window, *ZAMVTU* reads only the weights corresponding to the spatial coordinates of each nonzero activation and performs the XNOR-popcount operations. As described in the previous paragraph, the coordinates of nonzero activations are determined by the sliding window pattern, and the sliding window pattern is determined according to the output spatial coordinates of the window. Therefore, *ZAMVTU* checks the sliding window pattern by using the output spatial coordinates and generates the weight memory addresses of the weights corresponding to the coordinates of nonzero activations of the pattern. Then, using these addresses, the weights are read from the weight memory, and XNOR operations are performed on the weights and input activations. In the network architecture of Fig. 10, there are only four patterns of the sliding window, and the weight memory addresses corresponding to each pattern are illustrated in Fig. 12a. The weight memory is a buffer with a S-bit width and has all the weight filters allocated to PE, and only one filter is shown in the figure. In the case of weight filter pattern 1, it corresponds to the sliding window pattern 1 (red box) in Fig. 10, and XNOR-popcount operations are performed on this sliding window and weight filter. For this, the weight memory address generator generates addresses in the order of 8 and 9. *ZAMVTU* reads the weights using the created addresses and performs XNOR operations in synchronization with the input activation stream. Deconvolution utilizes flipped weight filters to perform convolution operations, and weight flip can be carried out off-line in advance. Alternatively, the weight filter flip can be processed on the fly without performing off-line in advance by creating the addresses in reverse order for each filter when generating the weight memory addresses. The partial sums resulting from the XNOR-popcounts are accumulated in the sliding window unit to obtain a convolution output, and batch normalization and sign activation are carried out by comparing the convolution output with the threshold calculated in consideration of zero activations. Thresholding considering zero activations is described in detail in IV-C.

Like *MVTU*, *ZAMVTU* processes input channels and weight filters in parallel to improve performance, but skipping zero activations does not destroy this parallelism. This is because since zero padding is performed based on the spatial direction, the input channels can still be processed in parallel

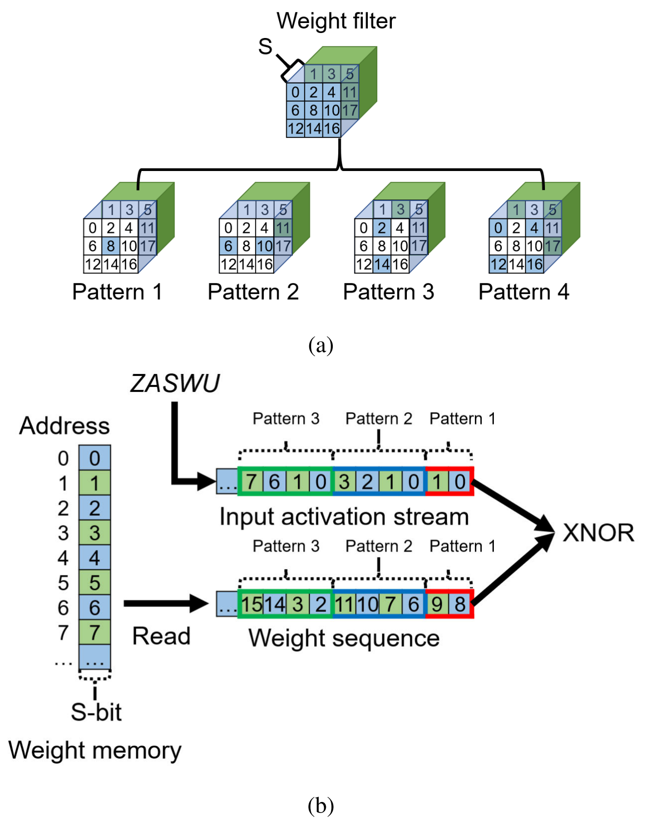

FIGURE 12. Zero-skipped convolution of *ZAMVTU*. (a) weight filter patterns, (b) *ZAMVTU* weight read and XNOR operation scheduling.

in the SIMD lanes because the padding is the same in the channel direction—for zero padded activations, all channels have a zero value. Also, since all PEs operate on the same input feature map, zero-skipped feature map can be processed in parallel in all PEs, and workloads of PEs are balanced.

There have been studies that skip the zero activation operations of *zero padding deconvolution*, as described above in II-D [21], [22], [38], [39]. GANAX [39] changed the dataflows on the 2-D systolic array to skip the operation of zero activations. In the baseline architecture they adopted, each PE of the systolic array performs vector processing on one filter row, and the dataflows are controlled so that different rows of the filter are processed in the x-axis direction and different output rows are generated in the y-axis direction of the array. Therefore, in this architecture, the PE rows handle different sliding windows. Since the zero patterns are different for each sliding window, the PE rows that process sliding windows of the same pattern are grouped into a PE group to apply SIMD, and the sliding windows of different patterns are processed by applying the multiple instruction multiple data (MIMD) method in different PE groups. Since the number of nonzero activations of sliding windows differs according to the pattern, when implemented only with SIMD, PEs handling sliding windows with more nonzero activations had to wait in idle state. By applying the SIMD-MIMD method, sliding windows of different patterns are handled separately in each PE group, so the idle cycles are eliminated and the utilization of PEs is improved. However, workload imbalance occurs between PE groups that handle different patterns, and this was not considered. In Song *et al.* [38], zero-skip was implemented by changing the dataflows and allocating nonzero activations and corresponding weights on the systolic array. This study proposed *zero free and output stationary (ZFOST)*, and each PE skips zeros to generate the output of specific spatial coordinates. Therefore, each PE handles one sliding window. As described above, since the number of nonzero activations between sliding windows is different due to zero padding, workload imbalance occurs between PEs, but this is not considered. On the other hand, in the proposed *ZAMVTU*, as described above, all PEs operate on the same nonzero input stream, so the workloads of all PEs are balanced.

In PAI-FCNN [22], sliding windows of various patterns due to zero padding are each processed by sub-convolution. Later, outputs of each sub-convolution were concatenated to create an output feature map through concatenation operations. Since this method processes sub-convolutions as a unit, its workloads are balanced, but it is a method of reconstructing dataflows with software in existing convolution hardware, different from our study in that ours changes dataflows with hardware. DT-CNN [21] implemented zero free deconvolution by adding delay cells to the systolic array. Convolution is carried out by partial sum aggregations while propagating the input feature maps that are not zero padded to the PE rows where the weights are stored. In the case of deconvolution, due to zero padding, it is necessary to accumulate the partial sums of different PE rows of different clock cycles, unlike conventional convolution. They solved this problem and implemented zero free deconvolution by adding delay cells between PE rows and latching partial sums into the delay cells. This study implemented deconvolution by reconstructing partial sum accumulations. Unlike this, our study implemented deconvolution by reconstructing the dataflows of input activations and weights.

C. ZERO-AWARE BATCH NORMALIZATION EMBEDDED BINARY ACTIVATION

FINN adopted as baseline architecture utilizes *BNEBA* mentioned in II-C, and processes batch normalization and sign activation as thresholding. Since the partial sums of XNOR results in convolution are obtained through unsigned popcounts, ranges of the convolution outputs of the sliding windows are changed from [$-fan\text{-}in$, $+fan\text{-}in$] to [0, +*fan-in*]. Therefore, in FINN, in order to match the ranges of the convolution outputs and the thresholds, shifting and scaling of thresholds are carried out as in (5). However, in *zero padding deconvolution*, the ranges of convolution outputs differ from the conventional convolution due to padded zeros. Since zero activations did not participate in accumulations, the ranges of convolution outputs are the number of nonzero activations excluding the number of zero activations. Therefore, in *zero padding deconvolution*, the shifting and scaling of the thresholds should also consider the ranges of the changed convolution outputs.

Fig. 13a shows the nonzero activations of each sliding window pattern in the network architecture of Fig. 10. If the number of input feature map channels is expressed as ICH, the nonzero activations of pattern 1, pattern 2 and 3, and pattern 4 are ICH, $2ICH$, and $4ICH$, respectively. Fig. 13b shows the ranges of convolution outputs and the ranges of thresholds calculated during training for each pattern of Fig. 13a. If the threshold calculated during training is th_{old} and the number of nonzero activations for each pattern is $fan\text{-}in_{pat}$, the range of th_{old} is $[-fan\text{-}in_{pat}, +fan\text{-}in_{pat}]$. In other words, th_{old} of patterns 1, 2 and 3, and 4 have ranges of $[-ICH, +ICH]$, $[-2ICH, +2ICH]$, $[-4ICH, +4ICH]$, respectively.

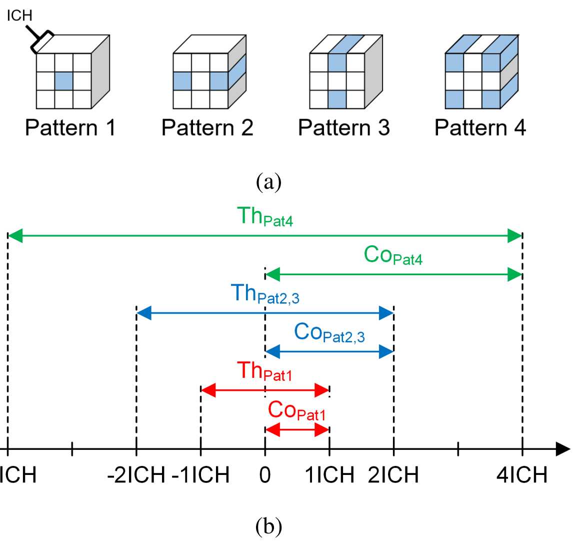

FIGURE 13. Range of convolution output and threshold in deconvolution. (a) nonzero activation (marked in blue) by sliding window pattern, (b) threshold and convolution output range by sliding window pattern, where Th, Co, Pat are threshold, convolution output, and pattern, respectively.

In order to fit the range of th_{old} to the convolution output ranges of each sliding window pattern, shifting and scaling must be performed using $fan\text{-}in_{pat}$ for each pattern. If the thresholds whose ranges are matched to the convolution outputs by sliding window pattern are called th_{new}, they are the same as (8).

$$th_{new} = (th_{old} + fan\text{-}in_{pat})/2 \tag{8}$$

The difference from (5) is that only fan-in has been changed to $fan\text{-}in_{pat}$, which is a fan-in for each pattern. Therefore, in order to calculate th_{new}, $fan\text{-}in_{pat}$ must be figured out. In the existing convolution, since the fan-in value is fixed for each network architecture, calculation of th_{new}, that is, threshold shifting and scaling can be carried out off-line in advance. Meanwhile, in *zero padding deconvolution*, $fan\text{-}in_{pat}$ is different in each sliding window pattern, so th_{new} cannot be precalculated off-line and it should be calculated during runtime. Therefore, to find th_{new}, *fan-in count unit* is added to

ZAMVTU to find out *fan-in*$_{pat}$ for each sliding window as illustrated in Fig. 14.

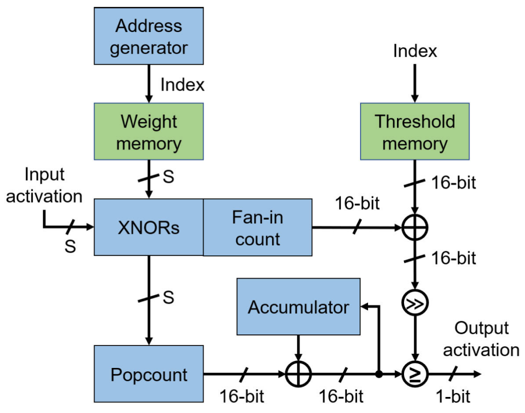

FIGURE 14. PE architecture of *ZAMVTU*.

Fig. 14 depicts the PE architecture of *ZAMVTU*. S input activations streamed from *ZASWU* and S weights read from the weight memory are computed in parallel in the XNOR array and transferred to the popcount unit. At this time, the addresses of the weight memory are generated by the address generator according to the spatial coordinates of the nonzero input activations. The popcount unit calculates the partial sums from the S XNOR results, and the calculated partial sums are accumulated in the sliding window unit in the accumulator to calculate the convolution outputs. The calculated convolution outputs are passed on to a comparator, and the comparator performs *BNEBA* and creates output activations by comparing the convolution outputs and th_{new} whose range is matched to the convolution outputs. As described above, th_{new} can be obtained by scaling and shifting th_{old} using *fan-in*$_{pat}$, a fan-in for each sliding window. *fan-in*$_{pat}$ is counted in *fan-in count unit*, and th_{old} calculated for *BNEBA* during training is stored in threshold memory. *fan-in count unit* increases fan-in by the number of computed activations each time XNOR operation of nonzero activations is performed. In other words, since S nonzero activations are operated in parallel in each PE, the fan-in is increased by S each time XNOR operation is carried out. The fan-in is counted for each sliding window and is initialized when the computations of the sliding window are completed. The counted fan-in value is added to th_{old} stored in the threshold memory, and then the added value is divided by 2 through the 1-bit right shift. Through this, shifting and scaling of (8) are carried out to obtain th_{new}. If the λ scale of (2) is negative, it is activated when the convolution output is smaller than the threshold ($Y \le th_{new}$). We change the direction of the inequality sign by inverting the sign of th_{old} and weights in order to utilize the greater-than threshold operation in this case as well.

BNEBA is not applied to the last convolution layer. This is because the outputs of the last convolution layer are scores so that the sign activation function should be skipped. In FINN, by skipping the thresholding of the last convolution layer, sign activation and batch normalization are not performed. However, batch normalization greatly affects the accuracy of the network, so not performing batch normalization leads to loss of accuracy. In GUINNESS [14], *BNEBA* is executed by adding the thresholds to the convolution outputs as a biases ($= -thresholds$) and comparing them with zero ($Y + bias \ge 0$). In this study, sign activation is skipped by performing only bias additions and not comparisons in the last convolution layer. However, biases do not include the λ scales of batch normalization, also leading to loss of inference accuracy. In the layers performing sign activation, the λ scales of batch normalization can be ignored because only the signs of the inputs are meaningful and the scales are not. Meanwhile, since the last convolution layer does not perform sign activation, it must be calculated including the λ scales to obtain the correct batch normalization results.

ZAMVTU of the last convolution layer performs batch normalization by subtracting thresholds from convolution outputs and multiplying the λ scales of batch normalization. Since the last convolution layer also uses unsigned popcount and zero edge padding, threshold shifting and scaling are performed considering *fan-in*$_{pat}$. The score calculation of the last convolution layer is as shown in (9).

$$\begin{aligned} score &= \frac{\lambda}{2}(Y - th_{new}) \\ &= \frac{\lambda}{2}\{Y - \frac{(th_{old} + fan\text{-}in_{pat})}{2}\} \end{aligned} \tag{9}$$

The λ scales use a 24-bit fixed point, and the λ scales have to be divided by 2 to match the range of th_{new}. However, dividing the λ scales by 2 does not affect the inference results, so we skip this in our design. This is because the inference is to find the class with the highest score in each spatial location, and all class scores are compared to each other in order to generate an inference result. Therefore, the operation of dividing by 2, which is applied equally to all class scores, is meaningless, and rather, there is a concern that loss of accuracy may occur due to lack of precision when dividing by 2. The λ scales of batch normalization can also be precalculated off-line like the thresholds. If the signs of the λ scales are negative, we have already considered this by inverting the signs of the thresholds and weights, so that the λ scales are calculated as absolute values.

Fig. 15 illustrates the PE structure of *ZAMVTU* for the last convolution layer. A 24-bit width scale memory is added for storing scales, and a subtractor for calculating the difference between the convolution outputs and th_{new} is added instead of a comparator for the activation function. And a 24-bit multiplier is added to multiply the scales and subtractions of the convolution outputs and the thresholds. Since *ZAMVTU* processes weight filters on multiple PEs in parallel, PEs can be configured up to the number of weight filters, and the number of λ scales to be stored is the same as the number of weight filters. The number of weight filters (i.e. the number of output channels) of the last convolution layer is the same as the number of classes, and generally, segmentation applications only

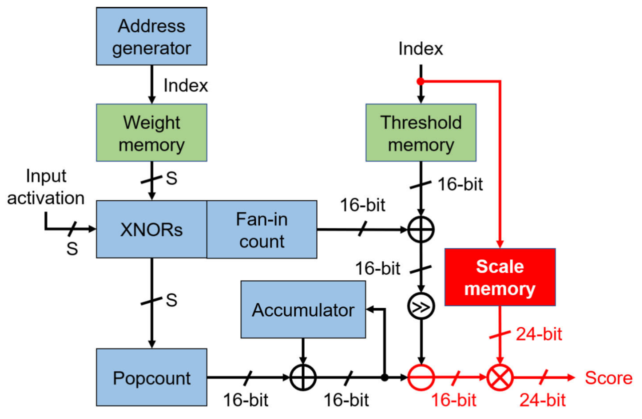

FIGURE 15. PE architecture of the last convolution layer's *ZAMVTU*.

require a classification for dozens of classes. Therefore, there are not many PEs and λ scales to be saved, so the hardware overhead for batch normalization of the last convolution layer is not large. For example, since the CamVid11 dataset has eleven classes, a maximum of eleven PEs can be configured to minimize the latency of the last convolution layer in the *BiDE* architecture. The only hardware overhead at this time is eleven 24-bit multipliers, eleven 16-bit subtractors, and eleven scale memories (registers in this case) that store each λ scale.

The first convolution layer performs partially binarized convolution where only the weights are binary. Since partially binarized convolution performs MACs, not XNOR-popcounts, the ranges of convolution outputs are the same as the ranges of thresholds. Therefore, the first convolution layer does not perform threshold shifting and scaling. *BNEBA* is executed by simply comparing the th_{old} calculated by (2) with the convolution outputs.

D. CONVOLUTION

BiDE adopts the convolution engine of the baseline architecture for convolution, and convolution is not the focus of this paper, so only the difference from the baseline will be briefly explained. Since semantic segmentation must classify all pixels of the input image, it does not transform the input image such as the crop. Also, since segmentation input images are generally rectangular, convolution for rectangular input images must be supported. Since the baseline architecture is a classification network accelerator, convolution for square input images is only supported. Therefore, in *BiDE*, it is modified to be able to perform convolution for rectangular inputs. In addition, the baseline architecture employs the max-pool. Since strided convolution is used to reduce the memory footprint in *BiDE* and *BEDN* as described above, the design is changed to support strided convolution using a stride of two. Finally, the baseline architecture employs a network structure in which the dimension of the feature maps gradually decreases due to the convolution that does not use zero padding on the edge. We change the design to support zero padding on the edge to maintain the spatial dimension of the feature maps. Since zero cannot be expressed in BNN, zero-skip is also applied to edge padding. There is a study that attempted +1 edge padding [48] to process edge padding in BNN. However, +1 edge padding adds unnecessary quantity, resulting in loss of accuracy. Since we employ zero edge padding, there is no loss in accuracy.

E. PIXEL-WISE CLASSIFICATION LAYER

The result of the last convolution layer is a score map that has the same spatial dimension as the input image and has as many channels as the number of classes, which is a segmentation result and is transferred to off-chip memory. The score map has scores of all classes for each spatial location, and each score is a real value, non-binary. To stream the score map to off-chip memory, bandwidth as much as *score map width* × *score map height* × *the number of classes* × *score bit width* × *FPS* is required. When the scores are expressed in 24-bit and the CamVid11 dataset is inferred at 30 FPS, a bandwidth of $480 \times 360 \times 11 \times 24 \times 30 = 1,368.576\ Mbits/s$, that is, $171.072\ MBytes/s$, is required. Such off-chip memory accesses adversely affect power consumption and performance. Therefore, we designed hardware that performs a pixel-wise classification layer to reduce the bandwidth required for transferring segmentation results to off-chip memory.

The pixel-wise classification layer is located after the last convolution layer and compares the scores of all classes by spatial locations to find the class index of the largest score. This layer finds class indices of all spatial locations and creates a class index map with the same size as the spatial dimension of the input image. The size of the class index map is *score map width* × *score map height* × *index bit width*, and the memory bandwidth requirement can be reduced by *the number of classes* × (*score bit width*/*index bit width*) compared to the score map. When the scores are expressed in 24-bit and inference of the CamVid11 dataset is performed, $11 \times 24/4 = 66$ times the memory bandwidth can be saved. The hardware performing the pixel-wise classification layer sequentially compares the scores of all classes at each spatial location, and spatial locations are processed in a pipelined manner. And then, the hardware streams only the class indices of the maximum scores of all spatial locations to the off-chip memory.

V. EXPERIMENTS

In this chapter, we evaluated the proposed *BEDN* and *BiDE*. First, the neural network model preparation and hardware accelerator implementation environment are described. After that, we analyzed the accuracy and memory footprint of *BEDN*, and then evaluated the performance, power consumption, and resource utilization of *BiDE* implemented on the FPGA.

A. ENVIRONMENT

We used the CamVid11 dataset, which is widely utilized in semantic segmentation performance evaluations, as a benchmark dataset to evaluate *BEDN*. Among the 701 images of the CamVid11 dataset, 421 images randomly extracted were used as a train set, and 280 others were utilized as a test set to train the network and measure the accuracy. Training of the network was carried out until no more increase in accuracy was observed. The models of *BEDN* employed in the experiment are shown in Table. 3. To check the loss of accuracy due to binarization, the performance of *BEDN* and the traditional FP segmentation networks FCN, DeeplabV3+, and SegNet were compared. To this end, the traditional networks also trained on the CamVid11 dataset. Also, to evaluate the *BEDN* model, SegNet was binarized in the same way as *BEDN*. FCN was initialized from the VGG-16, and we adopted a structure that upsamples the feature map of the last layer by 8 times. In DeepLabV3+, ResNet-18 was employed as the baseline architecture, and the stride for downsampling was 16. SegNet and Binarized SegNet were also preinitialized and trained based on the VGG-16. The input image size used in the experiment was 480 × 360. FCN, DeepLabV3+, SegNet, and *BEDN* were trained with an initial learning rate of 0.01, batch size of 12, and stochastic gradient descent with momentum (SGDM). Binarized SegNet was trained with an initial learning rate of 0.0001, batch size of 12, and adaptive moment estimation (ADAM).

BiDE was implemented on a ZCU104 board equipped with Xilinx XCZU7EV FPGA. The design and register transfer level (RTL) generation of *BiDE* were performed using Xilinx Vivado high-level synthesis (HLS), and the design was synthesized and implemented using Xilinx Vivado. The power consumption of *BiDE* was estimated using the Xilinx Vivado Power Estimator. *BiDE* is a scalable architecture, and the shape of the compute array performing each layer can be adjusted by the number of SIMD lanes S and the number of PEs P, which is an attribute inherited from the baseline architecture. In order to evaluate scalability, *BiDE* with three S, P configurations as Table. 4 were used in the experiment. Compared to the base configuration, the double configuration has twice as many SIMD lanes and the quad configuration has four times as many SIMD lanes as the base configuration. Each configuration is determined based on the number of operations in Table. 3 to match the latency of all layers except 1 and 11 layers equally to maximize the throughput of the pipeline. The source codes of proposed *BEDN* and *BiDE* are opened on GitHub. (https://github.com/IntelligenceDatum/BEDN, https://github.com/HyunwooKim2/BiDE.)

B. SEGMENTATION RESULT

We first discuss the performance of *BEDN*. The accuracy of the conventional FP segmentation network and *BEDN* for the CamVid11 dataset is shown in Table. 5 as intersection over union (IoU). Sky and road are large objects with a lot of training data, and even in *BEDN*, they showed high IoU at 91.2% and 91.2%, respectively. Sky and road showed only 4.0% and 6.6% of IoU drop compared to the average of FP networks, respectively. Sign-symbol, pedestrian, and column-pole showed low IoU at 36.9%, 31.2% and 24.2%, respectively. These classes have little training data and are small objects. *BEDN* training, like other quantization-aware training techniques, requires more epochs because its convergence is slower than that of the FP network model, and is sensitive to network architecture. For this reason, *BEDN* has a lower performance for objects with less training data. mIoU loss of *BEDN* was 8.0–12.5% compared to the FP network. We also applied the same binarization technique to other networks and compared their accuracy with *BEDN*. However, the binarized FCN and DeepLabV3+ did not get good accuracy and were excluded from the comparison. Compared with Binarized SegNet, *BEDN* showed 0.6% higher mIoU, and the accuracy difference between the two networks was not significant. Although SegNet has max-unpool and more layers, *BEDN* has a slightly better accuracy.

Fig. 16 shows samples of CamVid11 test set results. Overall, *BEDN* was able to distinguish each object well, and it showed better performance in segmenting small units than large units. Particles of other classes that were erroneously detected in small units can be seen scattered. Since *BEDN* does not use the original image or high-dimensional features in the decoding process, it can be confirmed that the objects were not cleanly filled. Section VI covers discussions on these topics.

TABLE 4. ***BiDE* configurations for *BEDN* inference.**

Layer	Base			Double			Quad		
	S	P	T_{CA}	S	P	T_{CA}	S	P	T_{CA}
1	3	16	48	3	32	96	3	64	192
2	32	32	1024	32	64	2048	64	64	4096
3	32	16	512	32	32	1024	32	64	2048
4	32	32	1024	32	64	2048	64	64	4096
5	32	16	512	32	32	1024	32	64	2048
6	32	32	1024	32	64	2048	64	64	4096
7	32	16	512	32	32	1024	32	64	2048
8	32	32	1024	32	64	2048	64	64	4096
9	32	16	512	32	32	1024	32	64	2048
10	32	32	1024	32	64	2048	64	64	4096
11	16	11	176	32	11	352	64	11	704
Total	-	-	7392	-	-	14784	-	-	29568

S = Number of SIMD lanes, P = number of PEs, T_{CA} = total SIMD lanes in compute array, Total = total SIMD lanes in *BiDE*.

C. MEMORY FOOTPRINT

This section discusses the memory footprint of *BEDN*. The network size and maximum memory usage of existing FP segmentation networks, Binarized SegNet and *BEDN*, are compared as shown in Table. 6. The network size refers to the size of the trained parameters composed of the weights of the convolution layers and batch normalization parameters. The batch normalization parameters of FP networks were calculated in consideration of folding into the convolution layer [44]. Besides, the memory usage for each layer was

TABLE 5. Performance comparison of networks for CamVid11 road class semantic segmentation.

Model	Buildings	Trees	Sky	Car	Sign-symbol	Road	Pedestrian	Fence	Column-pole	Side-walk	Bicyclist	mIoU
FCN-VGG16 [1]	94.7	88.7	96.3	88.4	50.3	98.0	53.6	57.4	22.1	90.2	71.8	67.9
SegNet-VGG16 [4]	88.0	80.8	92.9	82.2	46.1	96.4	43.4	63.6	30.5	83.6	64.9	70.2
DeepLabV3+-ResNet18 [5]	95.9	89.6	96.5	89.7	54.7	98.9	67.2	63.4	43.4	89.3	79.0	72.4
Binarized SegNet-VGG16	76.6	68.1	89.8	68.3	34.0	91.8	34.9	47.4	19.9	69.6	51.4	59.3
BEDN (Ours)	74.1	75.0	91.2	65.6	36.9	91.2	31.2	46.6	24.4	70.9	51.6	59.9

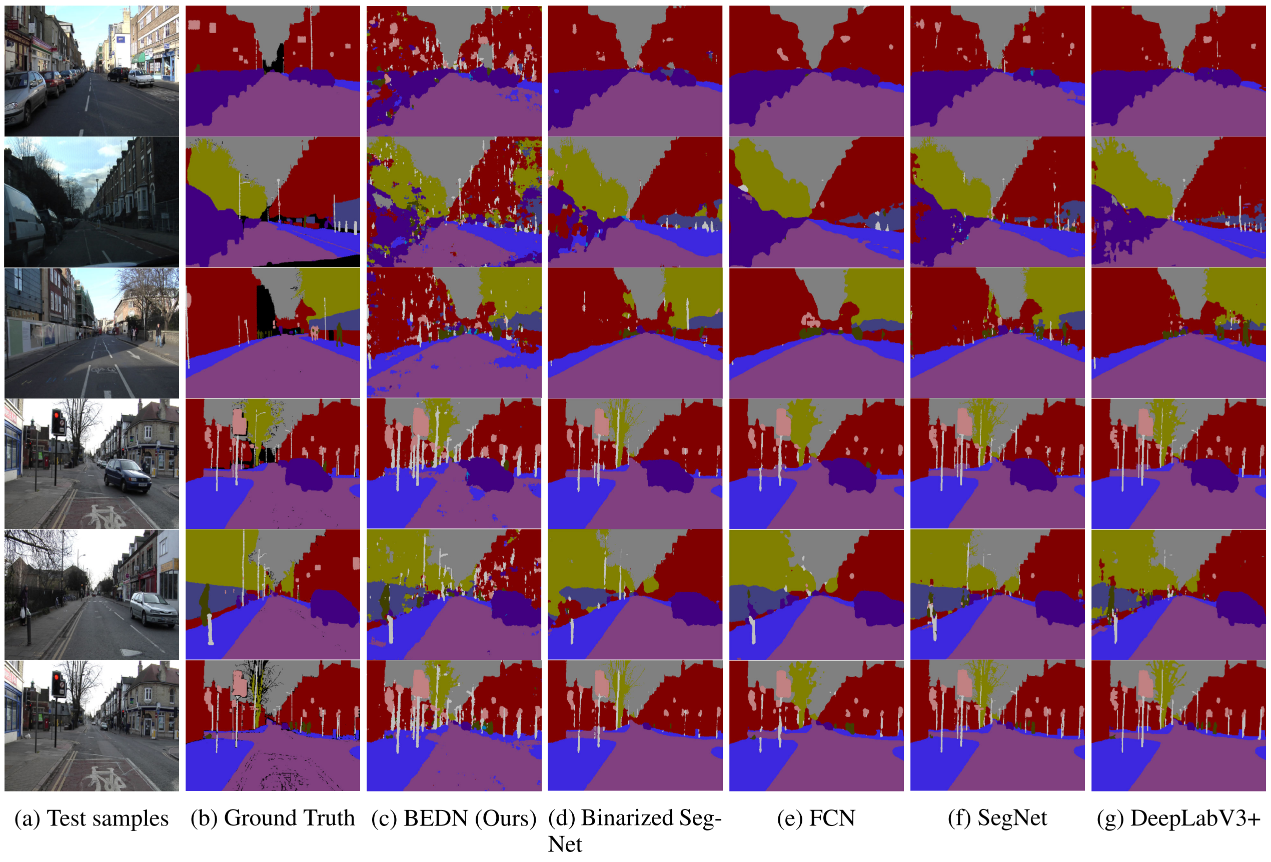

FIGURE 16. Samples of CamVid11 test-set results. (Best viewed in colors).

calculated, and the largest usage is shown in the maximum memory usage. The memory usage of each layer is the sum of the size of each layer's parameters, the size of the feature map that is the result of convolution, and the size of feature maps or max-pool indices waiting for calculation due to bypass connection. The parameter size of FP networks was calculated as 32-bit, the parameter size of Binarized SegNet and *BEDN* were calculated as 1-bit, and the max-pool index size of SegNet was calculated as 2-bit.

FCN had the largest size of the network and maximum memory usage due to the layer using 4096 7×7 filters. Since the architecture of SegNet was based on VGGNet, the size of the network and maximum memory usage were relatively large due to the deep channel of the layers. In SegNet inference, max-pool indices should be kept in memory until they are consumed by max-unpooling layer of the decoder. However, since the index was 2-bit, the size of max-pool indices was negligibly small compared to 32-bit parameters and feature maps, leading to a small effect on memory usage. DeepLabV3+ utilized ResNet for the encoder, and this was also based on VGGNet, so the size of the network was large. The maximum memory usage was the smallest among FP networks because downsampling at the front layer of the network greatly reduced the spatial dimension of the feature map. Compared to *BEDN*, the network size of Binarized SegNet was much larger despite being binary, because SegNet had many layers with a deep channel (512) and the number of layers was more than twice of that of *BEDN*. The maximum

TABLE 6. Memory footprint comparison of segmentation network models. (Unit: MB).

Model	Network size	Maximum memory usage
FCN-VGG16	537.35	416.14
SegNet-VGG16	117.70	44.93
DeepLabV3+-ResNet18	84.74	28.35
Binarized SegNet-VGG16	3.67	2.07
BEDN (Ours)	0.21	1.39

memory usage was also larger in Binarized SegNet because the feature map size was the same, but 691.2 KB was added due to the index map. In the case of the FP network, the 2-bit index map was negligible, but in the the case of binary network, the index map was also a large overhead. Since *BEDN* stored data in 1-bit and did not use feature map concatenation or max-unpool, the network size and maximum memory usage were very small compared to the FP networks. *BEDN* reduced network size by 397× and maximum memory usage by 20.43× compared to DeepLabV3+. Compared with Binarized SegNet, *BEDN* was structurally small, reducing the network size by 17×, and reducing the maximum memory usage by 1.49× by not using max-unpool.

D. PERFORMANCE AND POWER CONSUMPTION

To measure the performance of *BiDE*, *BEDN* was operated on *BiDE*, and inferences of 480 × 360 images of the CamVid11 test set were carried out 500 times. First, inferences were performed on each of the base, double, and quad configurations to check the scalability of *BiDE*, and the results are shown in Table. 7. GOPS was calculated based on the logical GOPs shown in Table. 3. As the number of SIMD lanes increased by 2× for double configuration and 4× for quad configuration, it can be seen that FPS and GOPS also increased at almost the same rate. On the other hand, the power consumption grew linearly, but to 1.53× and 2.5× less than the rate at which the SIMD lanes increased. This is due to the fact that resource usage increased less than the number of SIMD lanes. Resource usage according to the number of SIMD lanes is covered in the V-E. It is noteworthy that as the number of SIMD lanes increased, the performance efficiency (GOPS/W) also improved. Since the FPGA employed in the experiment cannot accommodate more SIMD lanes, the experiment requiring more SIMD lanes could not be conducted, but higher performance efficiency is expected in a larger FPGA.

TABLE 7. Performance and power consumption by configuration.

Configuration		Base	Double	Quad
Performance	FPS	6.58	13.12	25.89
	GOPS	428	854	1,685
Power Estimation (W)	Clocks	0.204	0.301	0.427
	Signals	0.189	0.32	0.593
	Logic	0.188	0.335	0.631
	BRAM	0.192	0.249	0.369
	DSP	0.046	0.044	0.025
	Total	0.819	1.249	2.045
Performance Efficiency (GOPS/W)		523	684	824

TABLE 8. Performance comparison with other implementation.

		Liu et al. [20] (2019)	BiDE (Ours)^a
Platform		Intel Arria 10	Xilinx ZCU7EV
Clock (MHz)		200	187.5
Bit precision		8-bit fixed	1-bit
Image Size		256x256	480x360
FPS		57	25.89
Power (W)		32	2
The number of OPs (GOPs)	L-GOPs	27.4	65
	P-GOPs	N/A	46
Performance	L-GOPS	1578	1682.85
	P-GOPS	N/A	1190.94
Performance Efficiency	FPS/W	1.78	12.95
	L-GOPS/W	49.3	824
	P-GOPS/W	N/A	595.47

^{a} Quad configuration

We also compared the performance of *BiDE* with Liu *et al.* [20], an existing 8-bit segmentation network accelerator, and the results are shown in Table. 8. *BiDE* carried out the inferences of 480 × 360 images of the CamVid11 dataset at 25.89 FPS with 2 watt (W) power consumption, and achieved 1682.85 logical GOPS and 1190.94 physical GOPS. Liu *et al.* accelerated the quantized U-Net to an 8-bit fixed point, inferred 256 × 256 size urban surface images at a rate of 57 FPS, and achieved 1578 logical GOPS. However, it consumed a large power of 32 W and exhibited a relatively low performance efficiency of 49.3 GOPS/W. *BiDE* showed a performance efficiency of 824 GOPS/W, achieving 16.7× higher performance compared to Liu *et al.* Moreover, even considering that Liu *et al.* was an 8-bit network accelerator, *BiDE* showed 2.1× higher performance efficiency. *BiDE* achieved higher performance than Liu *et al.* even in GOPS excluding power consumption.

E. RESOURCE UTILIZATION

We implemented *BiDE* on the FPGA, so we analyzed the FPGA resource utilization of *BiDE*. Fig. 17 shows the FPGA resource utilization for each *BiDE* configuration. The blue,

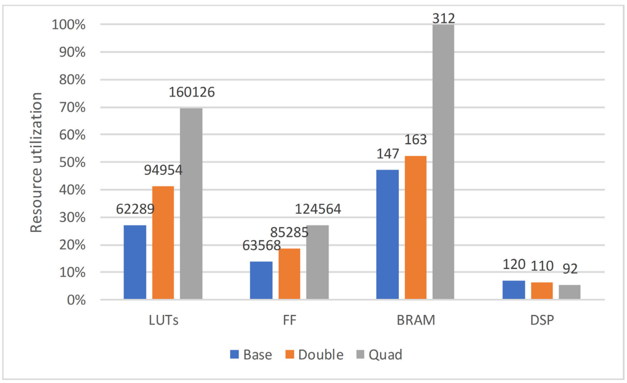

FIGURE 17. Resource utilization for each *BiDE* configuration.

orange, and gray bars represent the resource usage in the base, double, and quad configurations, respectively, and the usage relative to the total resources available in the FPGA is shown as a percentage on the vertical axis. The number at the top of each bar represents the actual number of resources utilized. As the number of SIMD lanes doubled and quadrupled, the resource usage increased. The lookup table (LUT), flip-flop (FF), and block random access memory (BRAM) usage increased by 1.52×, 1.34×, and 1.1×, respectively, in the double configuration, and increased by 2.57×, 1.96×, and 2.12×, respectively, in the quad configuration compared to the base configuration. The reason why resource usage did not increase as much as the number of SIMD lanes increased is that *ZASWU*, control logic, and on-chip memory size hardly grew even if the number of SIMD lanes increased.

The number of BRAMs increased less in double configuration than in base configuration, because many of the parameter memories in the double configuration were implemented as logic (LUTs and FFs). Among the 491 parameter memories, 320 (layers 3–9) were implemented as BRAM and 171 (layers 1–2, 10–11) were implemented as logic. In the base configuration, only 27 (1, 11 layers) of 251 parameter memories were implemented as logic. In the double configuration, a large part of the parameter memories was implemented as logic because the number of PEs was increased. As the number of PEs increased, the parameter size to be stored in the on-chip memory remained the same, but since more parameter memories were distributed, the size of each memory decreased. Since the number and size of the BRAM resources of the FPGA were limited, it was efficient to implement small memories as logic and only large memories as BRAM. Layers 1, 2, 10, and 11 had a small size of input channels and output channels and thus a small size of parameters, so the parameter memories were implemented as logic. In the case of quad configuration, the number of PEs was further increased and the parameter memories were further divided and distributed. Even though 100% of BRAM resources were used, the number of BRAMs was insufficient, and the rest were implemented as logic. Furthermore, in the case of quad configuration, 70% of the LUTs were utilized, but it was the maximum configuration that could be implemented on the FPGA employed in the experiments because more logic could not be used due to routing problems.

In Table. 9, the FPGA resource utilization of the quad configuration of *BiDE* and that of Liu *et al.* [20] are compared. *BiDE* and Liu *et al.* utilized similar amounts of FFs and BRAMs. However, while Liu *et al.* required off-chip memory to store feature maps, our design streamed feature maps, so segmentation inference could be performed with only on-chip memory. *BiDE* used 1.87× more LUTs than Liu *et al.*, and Liu *et al.* utilized 6.96× more digital signal processing slices (DSPs) than *BiDE*. This is because Liu *et al.* was an 8-bit network accelerator, using MAC operations for convolution and thus DSPs were used to implement convolution. On the other hand, since *BiDE* used XNOR-popcount operations for convolution, LUTs were utilized to implement convolution. Although *BiDE* implemented the convolution operations only with LUTs due to binary operations and data representations, the level of LUT usage was similar to that of Liu *et al.*, which did not utilize LUTs to implement convolution.

TABLE 9. Comparison of resource utilization with conventional segmentation accelerator.

Resource	LUT	FF	BRAM	DSP
Liu et al. [20]	85,679	110,643	364	640
BiDE (Ours)[a]	160,126	124,564	312	92

[a] Quad configuration.

VI. DISCUSSION

Like other networks, *BEDN* can set a trade-off between accuracy and network size through the proposed unit repetition. *BEDN* focuses on reducing memory usage through binary-quantization and simple structure in consideration of the hardware architecture and constraints, so it can be seen that the accuracy is lower than that of the existing FP-based network. In particular, *BEDN* and *BiDE* have cross-dependency with each other. *BEDN* designed for maximum hardware performance has several shortcomings due to its structural limitations of the network. First, as in some cases shown in Fig. 16, *BEDN* does not provide clean filling when segmenting large objects, and many false positives occur when segmenting small objects. This problem arises because the structure of the model is serial, so the original image or high-dimensional features cannot be utilized with bypass connection or concatenation. Secondly, *BEDN* cannot have a large and complex network structure. This means that in the field of image segmentation, more extensive research on the training method and model structure of BNN along with hardware constraints is required. Based on the bad results of simply applying binarization to DeepLabV3+ and FCN, the structure of other well-known segmentation models cannot be simply applied to overcome the shortcomings of *BEDN*. Therefore, model structure exploration and binary-quantization for complex segmentation models are our future work.

Since *BiDE* is based on a heterogeneous streaming architecture and has compute arrays for each layer, the hardware size linearly increases according to the number of layers in the network. Therefore, there is a limit to the increase in the number of layers of the network due to the hardware structure. If there are many layers, hardware size can be reduced by reducing the number of SIMD lanes in each compute array, that is, logic size, but latency increases. In other words, there is a trade-off among the number of layers, logic size, and latency. And the proposed *zero-aware binary deconvolution* skips operation and storage of zero activation according to the sliding window pattern. When the stride is 2, the number of patterns is small (4), so the hardware overhead is not large. However, as the stride increases, the number of patterns increases, thus increasing the hardware complexity and overhead. This means there is a trade-off between the number of patterns and the hardware overhead. Experiments have shown that increasing the number of SIMD lanes in *BiDE* increases hardware size and power consumption, and also increases performance at a larger rate. Therefore, in *BiDE* architecture, hardware size, power consumption, and performance can be balanced by adjusting the number of SIMD lanes.

VII. CONCLUSION

The semantic segmentation network has a large computation and memory requirement, so it is difficult to operate in an environment where the power budget is limited and hardware resources are insufficient. In this paper, we propose and design *binarized encoder-decoder network (BEDN)* and *binarized deconvolution engine (BiDE)* to enable low-power, real time semantic segmentation.

The designed *BEDN* considers memory usage, computational demand, and hardware constraints for a far-edge environment. For training of *BEDN*, a binary-quantization-aware training method of an encoder-decoder structure was organized. Through the binary-quantization and a repetitive unit structure, *BEDN* reduced network size by 397× and maximum memory usage by 20.43× compared to DeepLabV3+ and achieved a mIoU of 59.9% for the CamVid11 dataset. The full-binarization of weights and activations and a simple structure that does not use feature map concatenation or max-unpool was achieved for *BiDE*. *BEDN* used up to 1.38 MB of memory, and the network size was only 0.21 MB.

BiDE targeted FPGA and was designed based on a heterogeneous streaming architecture, and accelerated the inference of *BEDN*, a binarized segmentation network. The streaming architecture and simple segmentation network structure allowed all feature maps to be processed on-chip, eliminating all costly off-chip memory accesses of feature maps. Moreover, in order to enable the deconvolution of a binarized segmentation network that cannot represent zero, *zero-aware binary deconvolution* that skips operations and storage of padded zero activations, and *zero-aware batch normalization embedded binary activation*, performing batch normalization and binary activation in consideration of zero activations, were introduced. *BiDE* was implemented to operate at 187.5 MHz using 160k LUTs, 125k FFs, and 312 BRAMs on Xilinx XCZU7EV FPGA. *BiDE* accelerated *BEDN* to perform 480 × 360 image inferences of the CamVid11 dataset at 25.89 FPS and achieved a performance of 1.68 TOPS and 824 GOPS/W. In addition, *BiDE* is a scalable architecture that can configure the number of PEs and the number of SIMD lanes in the PE. As the total number of SIMD lanes increased, the performance efficiency, that is, OPS/W, enhanced in line. In the experiment, when the number of SIMD lanes increased to 2× and 4×, the performance efficiency improved by 1.3× and 1.57×. In this paper, the saturation point of the improvement of performance efficiency could not be identified because the SIMD lanes could not be increased any more due to the resource limit of the FPGA, but it is expected that higher performance efficiency can be confirmed in a larger FPGA.

ACKNOWLEDGMENT

Hyunwoo Kim was in charge of the development of hardware accelerator (*BiDE*) and Jeong Hoon Kim was in charge of network design and training (*BEDN*). The authors would like to thank the anonymous reviewers for their valuable feedback and comments.

REFERENCES

[1] 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.

[2] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1520–1528.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.

[4] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[5] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proc. ECCV*, 2018, pp. 801–818.

[6] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi, "ESPNet: Efficient spatial pyramid of dilated convolutions for semantic segmentation," in *Proc. ECCV*, 2018, pp. 552–568.

[7] R. P. K. Poudel, S. Liwicki, and R. Cipolla, "Fast-SCNN: Fast semantic segmentation network," 2019, *arXiv:1902.04502*. [Online]. Available: http://arxiv.org/abs/1902.04502

[8] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," 2016, *arXiv:1606.02147*. [Online]. Available: http://arxiv.org/abs/1606.02147

[9] M. Treml, J. Arjona-Medina, and T. Unterthiner, "Speeding up semantic segmentation for autonomous driving," in *Proc. MLITS, NIPS Workshop*, vol. 2, 2016, p. 7.

[10] M. AskariHemmat, S. Honari, L. Rouhier, C. S. Perone, J. Cohen-Adad, Y. Savaria, and J.-P. David, "U-net fixed-point quantization for medical image segmentation," in *Proc. LABELS/HAL-MICCAI/CuRIOUS*, vol. 2019, pp. 115–124.

[11] X. Xu, Q. Lu, L. Yang, S. Hu, D. Chen, Y. Hu, and Y. Shi, "Quantization of fully convolutional networks for accurate biomedical image segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8300–8308.

[12] H. Wen, S. Zhou, Z. Liang, Y. Zhang, D. Feng, X. Zhou, and C. Yao, "Training bit fully convolutional network for fast semantic segmentation," 2016, *arXiv:1612.00212*. [Online]. Available: http://arxiv.org/abs/1612.00212

[13] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, Jan. 2018.

[14] 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.

[15] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. FPGA*, 2017, pp. 65–74.

[16] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 w," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, Apr. 2018.

[17] A. Al Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini, "XNORBIN: A 95 TOp/s/W hardware accelerator for binary convolutional neural networks," in *Proc. IEEE Symp. Low-Power High-Speed Chips (COOL CHIPS)*, Apr. 2018, pp. 1–3.

[18] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2016, pp. 77–84.

[19] S. Liu, H. Fan, X. Niu, H.-C. Ng, Y. Chu, and W. Luk, "Optimizing CNN-based segmentation with deeply customized convolutional and deconvolutional architectures on FPGA," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–22, Dec. 2018.

[20] S. Liu and W. Luk, "Towards an efficient accelerator for DNN-based remote sensing image segmentation on FPGAs," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 187–193.

[21] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, "DT-CNN: Dilated and transposed convolution neural network accelerator for real-time image segmentation on mobile devices," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[22] L. Xia, L. Diao, Z. Jiang, H. Liang, K. Chen, L. Ding, S. Dou, Z. Su, M. Sun, J. Zhang, and W. Lin, "PAI-FCNN: FPGA based inference system for complex CNN models," in *Proc. IEEE 30th Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2019, pp. 107–114.

[23] D. Xu, K. Tu, Y. Wang, C. Liu, B. He, and H. Li, "FCN-engine: Accelerating deconvolutional layers in classic cnn processors," in *Proc. ICCAD*, 2018, pp. 1–6.

[24] Z. Li, B. Li, Z. Fan, and H. Li, "RED: A ReRAM-based efficient accelerator for deconvolutional computation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4736–4747, Dec. 2020.

[25] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognit. Lett.*, vol. 30, no. 2, pp. 88–97, Jan. 2009.

[26] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[27] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the Kitti vision benchmark suite," in *Proc. CVPR*, Jun. 2012, pp. 3354–3361.

[28] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. NIPS*, 2016, pp. 4107–4115.

[29] S. Darabi, M. Belbahri, M. Courbariaux, and V. Partovi Nia, "Regularized binary network training," 2018, *arXiv:1812.11800*. [Online]. Available: http://arxiv.org/abs/1812.11800

[30] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. NIPS*, 2015, pp. 3123–3131.

[31] L. Hou, Q. Yao, and J. T. Kwok, "Loss-aware binarization of deep networks," 2016, *arXiv:1611.01600*. [Online]. Available: http://arxiv.org/abs/1611.01600

[32] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. ECCV*, 2016, pp. 525–542.

[33] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-real net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm," in *Proc. ECCV*, 2018, pp. 722–737.

[34] D. J. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and H. P. Leong, "High performance binary neural networks on the Xeon+FPGA platform," in *Proc. 27th Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2017, pp. 1–4.

[35] L. Yang, Z. He, and D. Fan, "A fully onchip binarized convolutional neural network FPGA impelmentation with accurate inference," in *Proc. Int. Symp. Low Power Electron. Design*, Jul. 2018, pp. 1–6.

[36] L. Yang, Y. Zhang, J. Chen, S. Zhang, and D. Z. Chen, "Suggestive annotation: A deep active learning framework for biomedical image segmentation," in *Proc. MICCAI*, 2017, pp. 399–407.

[37] D. Wang, J. Shen, M. Wen, and C. Zhang, "Efficient implementation of 2D and 3D sparse deconvolutional neural networks with a uniform architecture on FPGAs," *Electronics*, vol. 8, no. 7, p. 803, Jul. 2019.

[38] M. Song, J. Zhang, H. Chen, and T. Li, "Towards efficient microarchitectural design for accelerating unsupervised GAN-based deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 66–77.

[39] A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "GANAX: A unified MIMD-SIMD acceleration for generative adversarial networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 650–661.

[40] H. Mao, M. Song, T. Li, Y. Dai, and J. Shu, "LerGAN: A zero-free, low data movement and PIM-based GAN architecture," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 669–681.

[41] J. Yan, S. Yin, F. Tu, L. Liu, and S. Wei, "GNA: Reconfigurable and efficient architecture for generative network acceleration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2519–2529, Nov. 2018.

[42] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *Proc. ECCV*, 2008, pp. 44–57.

[43] L. Yang, Z. He, and D. Fan, "A fully onchip binarized convolutional neural network FPGA impelmentation with accurate inference," in *Proc. Int. Symp. Low Power Electron. Design*, Jul. 2018, pp. 1–6.

[44] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "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.

[45] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[46] S. Venkataramani, V. Srinivasan, J. Choi, P. Heidelberger, L. Chang, and K. Gopalakrishnan, "Memory and interconnect optimizations for peta-scale deep learning systems," in *Proc. IEEE 26th Int. Conf. High Perform. Comput., Data, Anal. (HiPC)*, Dec. 2019, pp. 225–234.

[47] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[48] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2017, pp. 15–24.

HYUNWOO KIM received the B.S. degree in electronic engineering from Dongeui University, Busan, South Korea, in 2011, and the M.S. degree in electronics and computer engineering from Hanyang University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree in electronics and computer engineering. His current research interests include neural network accelerator, neural processing unit, and system on-chip.

JEONGHOON KIM received the B.S. degree in automotive engineering from Kookmin University, Seoul, South Korea, in 2015, and the M.S. degree in electrical engineering from Korea University, Seoul, in 2018. He is currently working with LG Chemical as a Specialist of Artificial Intelligence Technology Team. His current research interests include neural network quantization, model compression, and robotics perception and its industrial applications.

JUNGWOOK CHOI (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2008 and 2010, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 2015. He worked with the IBM Thomas J. Watson Research Center as a Research Staff Member, from 2015 to 2019. He is currently an Assistant Professor with the Department of Electronic Engineering, Hanyang University, Seoul. His current research interest includes efficient implementation of deep learning algorithms. He has actively contributed to the academic activities, such as Technical Program Committee from 2018 to 2020 (Co-Chair) and DAC 2018–2020, and Technical Committee (DiSPS) in the IEEE Signal Processing Society. He has received several research awards, such as the DAC 2018 Best Paper Award.

JUNGKEOL LEE (Student Member, IEEE) received the B.S. degree in electronic engineering from Hanyang University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree in electronics and computer engineering. His current research interests include embedded computing and the IoT device.

YONG HO SONG (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 1989 and 1991, respectively, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 2002. He is currently a Professor with the Department of Electronic Engineering, Hanyang University, Seoul, and a Senior Vice President with Samsung Electronics Company Ltd. His current research interests include system architecture and software systems of mobile embedded systems that further include SoC, NoC, multimedia on multicore parallel architecture, and NAND flash-based storage systems. He served as a Program Committee Member for several prestigious conferences, including the IEEE International Parallel and Distributed Processing Symposium, the IEEE International Conference on Parallel and Distributed Systems, and the IEEE International Conference on Computing, Communication, and Networks.