A Real-Time Object Detection Processor With XNOR-Based Variable-Precision Computing Unit

Wonjae Lee, Kukbyung Kim, Woohyun Ahn, Jinho Kim, and Dongsuk Jeon, *Member, IEEE*

***Abstract*— Object detection algorithms using convolutional neural networks (CNNs) achieve high detection accuracy, but it is challenging to realize real-time object detection due to their high computational complexity, especially on resource-constrained mobile platforms. In this article, we propose an algorithm-hardware co-optimization approach to designing a real-time object detection system. We first develop a compact object detection model based on a binarized neural network (BNN), which employs a new layer structure, the DenseToRes layer, to mitigate information loss due to deep quantization. We also propose an efficient object detection processor that runs object detection with high throughput using limited hardware rescources. We develop a resource-efficient processing unit supporting variable precision with minimal hardware overheads. Implemented in field-programmable gate array (FPGA), the object detection processor achieves 64.51 frames/s throughput with 64.92 mean average precision (mAP) accuracy. Compared to prior FPGA-based designs for object detection, our design achieves high throughput with competitive accuracy and lower hardware implementation costs.**

***Index Terms*— Binarized neural network (BNN), convolutional neural network (CNN), field-programmable gate array (FPGA), low-power design, object detection.**

I. Introduction

RECENTLY, we have seen rapid advances in computer vision, enabled by adopting deep learning approaches. Various deep learning models offered very high algorithm accuracy even comparable to that of human beings, which opens the possibility of using those algorithms in practical applications. Object detection, a widely studied sub-category of computer vision, is a process of finding specific objects in the input image or video [1], [2], [3], [4]. Typically, the algorithm produces the location of each object in the image along with its object type. It is an essential element in various applications such as autonomous driving, security, and robotics [5]. However, deep learning algorithms typically require a large amount of computation and memory access, making it difficult to implement those algorithms on hardware.

Manuscript received 20 September 2022; revised 7 February 2023; accepted 28 February 2023. Date of publication 21 March 2023; date of current version 23 May 2023. This work was supported by a grant-in-aid of Hanwha Systems. *(Wonjae Lee and Kukbyung Kim contributed equally to this work.) (Corresponding author: Dongsuk Jeon.)*

Wonjae Lee, Kukbyung Kim, and Dongsuk Jeon are with the Graduate School of Convergence Science and Technology, the Research Institute for Convergence Science, and the Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, South Korea (e-mail: djeon1@snu.ac.kr).

Woohyun Ahn and Jinho Kim are with Hanwha Systems, Seongnam 13524, South Korea.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2023.3257198.

Digital Object Identifier 10.1109/TVLSI.2023.3257198

To address this issue, a wide range of hardware accelerators for deep learning models have been proposed. Most deep learning models employ deep neural networks such as convolutional neural networks (CNNs), and they are usually realized using matrix multiplications. Hence, deep learning accelerators are aimed at efficiently accelerating matrix multiplication through parallelization and architecture optimization to maximize throughput [6]. However, the computational complexity of deep learning models for object detection continues to increase [7], [8], [9], and the amount of computation and memory bandwidth well exceeds those available on embedded platforms. Therefore, to achieve real-time object detection on embedded platforms with limited hardware resources and power budget, we need to develop a highly optimized deep neural network accelerator paired with a lightweight object detection algorithm having high detection accuracy.

Deep learning model optimization is an active field of research aimed at reducing the computational complexity and memory requirements of deep learning models while preserving algorithm accuracy. For instance, quantization [10], pruning [11], and compression [12] are often employed to reduce the number of model parameters and required operations. Model quantization reduces the number of bits that represent model parameters such as synaptic weights and biases. Since the area and power consumption of arithmetic units decrease with the bitwidth linearly (e.g., adders) or quadratically (e.g., multipliers) [13], model quantization effectively improves computation power efficiency and reduces hardware overhead, which is critical in embedded applications. In addition, reducing the model size through quantization also decreases the amount of external memory access that dissipates significantly larger energy than ON-chip computation [14]. Furthermore, recent studies show that even binarized neural networks (BNNs) achieve good algorithm accuracy comparable to full-precision models in various tasks including image classification and object detection [15], [16], [17], [18], [19], [20]. In BNNs, the synaptic weights and activations are expressed as 1-bit values, significantly reducing memory bandwidth and hardware resources. However, using fewer bits often leads to noticeable degradation in algorithm accuracy (e.g., mean average precision (mAP) in object detection), and a range of algorithmic techniques have been proposed to mitigate this issue.

Recent studies present various application-specific integrated circuit (ASIC) implementations to fully utilize the advantage of deep quantization [21], [22], [23], [24]. Ando et al. [21] propose an efficient binary/ternary neural

network accelerator with an in-memory computing scheme. On the other hand, the design in [23] supports variable weight bit precision to support various quantized neural networks. The high degree of freedom of ASIC allows designers to apply hardware design techniques to optimize the hardware for the target application and model. Hence, ASIC is usually considered the most power-efficient solution when it comes to accelerating BNN models. However, the initial design and prototyping costs for ASIC are huge, especially when fabricated in advanced nodes, and hence it is not an ideal solution for applications with low-volume production. Also, due to complex design verification and fabrication processes, it is challenging to reduce time-to-market.

Field-programmable gate array (FPGA) is increasingly utilized for accelerating deep learning as it offers an optimal design choice in the trade-off between hardware flexibility and processing efficiency. Since we can reconfigure the hardware at the gate level in FPGA, the same hardware design techniques could be applied to maximize performance and efficiency. In addition, the prototyping and production costs are significantly lower than those of ASIC, which makes FPGA an attractive design choice in embedded applications.

In this work, we present an FPGA-based processor for real-time object detection on embedded platforms. We first propose a deeply quantized neural network model for object detection. We minimize performance degradation while binarizing weights and activations in most layers throughout the network by proposing quantization-friendly layer structures. Then, we present an object detection processor supporting the proposed object detection algorithm efficiently. The processor features a resource-efficient variable bit-precision processing element (PE) and completely removes OFF-chip DRAM access by storing the entire model in ON-chip memory. Our design demonstrates real-time object detection with 64.51 frames/s throughput and 64.92 mAP and consumes 6.58 W.

The rest of the article is organized as follows. Section II summarizes the background and related work. Section III details the proposed object detection algorithm and Section IV discusses the processor architecture. Experimental results are presented in Section V. Finally, Section VI concludes the article.

II. Related Work and Background

A. Related Work on BNN

As BNNs use binary weights and binary activations, external memory bandwidth is greatly reduced and convolutional layers are efficiently implemented using simple XNOR and bit count operations. An early work in [25] demonstrates a BNN that adopts GoogleNet [26], and it achieves 47.1% top-1 accuracy on ImageNet [27]. The XNOR-Net [15] shows an improved performance of 51.2% top-1 accuracy for ImageNet classification by introducing scaling factors to reduce quantization errors and reordering the sequence of layers for effective training. In addition, the authors suggest that multiplication between activation and weight can be efficiently implemented using XNOR operations in binarized layers. However, some layers (e.g., first and last layers) are not binarized to avoid performance drops, which necessitates separate arithmetic units to support high-precision operations and decreases hardware utilization. The Bi-Real Net [16] utilizes additional shortcut connections to compensate for information loss due to deep quantization, along with a piecewise linear approximation of the derivative of the sign function. The authors also propose a magnitude-aware gradient and a clip function to replace the ReLU function. The Bi-Real Net achieves 56.4% and 62.2% top-1 accuracy on ImageNet with 18 and 34 layers, respectively. The real-to-binary convolutions [17] adopts the block structure of the XNOR-Net, double-skip connections, and real-valued downsample layers of the Bi-Real Net. It also introduces trainable scaling factors, which results in 65.4% top-1 accuracy on ImageNet.

B. Related Work on FPGA-Based Object Detection Processor

Recent studies on FPGA-based object detection hardware employ deep quantization to enhance processing efficiency. Nakahara et al. [28] propose Lightweight YOLOv2 and demonstrate the first FPGA object detector using BNN. However, since binarizing the model severely degrades the algorithm accuracy, they adopt a full-precision parallel support vector regressor for localization, which significantly increases external memory access. Tincy you-only-look-once (YOLO) [29] employs 1-bit weights and 3-bit activations in the entire neural network. The authors report the end-to-end processing time from capturing an image to displaying detection results, but their design processes the first and last layers on software as they severely limit throughput. Sim-YOLO-v2 [30] uses 1-bit weights and 4–6-bit activations in the neural network. The authors propose a streaming accelerator architecture that reduces external DRAM access to improve throughput and power efficiency. CoDeNet [31] adopts input-adaptive deformable convolution to develop an efficient network for object detection, and the network is quantized with 4-bit weights and 8-bit activations. The authors propose hardware-friendly modification of a deformable convolution operation on FPGA. The real-time SSDLite [32] adopts MobileNetV2 [33] as a backbone network for object detection. This model is not deeply quantized, but the design demonstrates the highest throughput to date. This is achieved through an efficient processing unit supporting both depthwise and pointwise convolutions and the task control unit that alleviates the overhead of job scheduling.

C. YOLO Object Detection Models

The YOLO models [4], [7], [8], [34] are representative one-stage object detector algorithms based on CNN, which predict the bounding box and class of objects simultaneously in the input image. They have a fast detection speed to enable real-time object detection, and the object detection accuracy has improved in more recent versions. YOLO models consist of two parts: a backbone network and a head network. The backbone network extracts features from the input image, and

the head network detects objects by performing localization and classification using extracted features. The backbone network is usually pretrained for the image classification task on a large dataset such as ImageNet. Then, the head network is attached to it, and the entire model is trained for the object detection task.

YOLOv2 [34] uses DarkNet-19, which is similar to VGG-19, as the backbone network and uses five convolution layers in the head network. The model divides the input image into $S \times S$ grids and predicts five bounding boxes for each grid based on anchor boxes. Each bounding box consists of location (x and y), size (w and h), confidence score, and classification score for 20 classes. Therefore, the detector outputs 125 values for each grid. Finally, post-processing such as non-maximum suppression (NMS) is performed to find the best candidate from the detected boxes and complete object detection. While more recent versions of YOLO models provide higher object detection accuracy, they utilize much larger backbone networks and complex connections between the backbone and head networks. Hence, the original YOLO and YOLOv2 have been widely used for ASIC [35], [36], and FPGA [28], [29], [30] implementations.

D. Model Quantization

The power consumption and area of arithmetic blocks are dictated by the precision they support, and hence model quantization is a key to efficient hardware implementation. In addition, we can significantly reduce memory footprint and bandwidth by quantizing data stored in external memory. Provided the number of bits n and quantization scale s, the quantized value of a real number x_r is expressed by

$$x_q = s \times \left(\text{round}\left(\frac{x_r}{s} + 0.5\right) - 0.5\right). \tag{1}$$

Note that the equation above represents a uniform and symmetric quantization scheme that does not include zero. Then, the maximum value which can be represented in this scheme is determined by

$$x_{q\,\text{max}} = s \times \left(\frac{2^n - 1}{2}\right). \tag{2}$$

If x_r is larger than $x_{q\,\text{max}}$, then its quantized value is clipped to $x_{q\,\text{max}}$.

In 1-bit convolution layers of typical BNNs, the weights and activations are both binarized to -1 and 1 through a sign function

$$x_b = \text{sign}(x_r) = \begin{cases} +1, & \text{if } x_r \geq 0 \\ -1, & \text{if } x_r < 0. \end{cases} \tag{3}$$

If -1 and 1 are expressed as 0 and 1 as shown above, multiplication and addition are replaced by bitwise XNOR and bit count operations

$$\mathbf{X_b} \cdot \mathbf{W_b} = 2 \cdot \text{BitCount}\left(\text{XNOR}\left(\hat{\mathbf{X}}_\mathbf{b}, \hat{\mathbf{W}}_\mathbf{b}\right)\right) - N \tag{4}$$

where $\mathbf{X_b}$ and $\mathbf{W_b}$ are the vectors of binary activations and binary weights represented as -1 or 1 with N elements, while $\hat{\mathbf{X}}_\mathbf{b}$ and $\hat{\mathbf{W}}_\mathbf{b}$ are identical vectors represented as 0 or 1. These operations can be efficiently implemented in hardware; for instance, look-up tables (LUTs) could be used to realize this function on FPGA. In addition, to reduce the quantization error of the binarized weights, the mean of the absolute weight values can be used as a scaling factor α

$$w_b = \alpha \cdot \text{sign}(w_r) = \begin{cases} +\alpha, & \text{if } w_r \geq 0 \\ -\alpha, & \text{if } w_r < 0. \end{cases} \tag{5}$$

This scaling factor is usually absorbed in the batch normalization layer during inference and does not increase the amount of computation [16], [18].

III. Binarized Object Detection Algorithm

Recent studies reveal that deep neural networks are relatively resilient to computation errors incurred during inference. Hence, various model optimization techniques such as quantization have been extensively studied, and commercial devices now provide optimized datapaths for low-precision operations [37], [38]. In this work, we propose a software–hardware co-optimization approach to design an energy-efficient object detection processor. In this section, we first propose a deeply quantized object detection model with a good algorithm accuracy and low hardware implementation costs. More specifically, we aim at minimizing the model size so that the entire model could be stored in on-chip memory. Then, the processor can avoid costly external memory access and achieves higher energy efficiency. In addition, representing weight and activation in low precision also reduces the power consumption of computing units.

A. Backbone Network Design

In object detection models consisting of a backbone network and a head network, the backbone network is usually pretrained for image classification tasks. Experimental results in Fig. 1 show that the object detection performance is highly correlated with the classification accuracy of the backbone network; the better the image classification performance of the backbone network, the better the object detection performance of the entire network. The backbone networks in YOLO models are generally prone to performance drop when deeply quantized. For instance, the backbone network of YOLOv2, DarkNet-19, has a similar structure to VGG-19, and it suffers large performance degradation when activations are quantized into less than three bits [30].

To alleviate this issue, we propose a new structure called DenseToRes layer shown in Fig. 3(a). The DenseToRes layer places both dense and residual connections in series. The layer first processes the input using dense connections to concatenate the input and output features. This preserves the undamaged information [19] as separate channels, but the preserved information is not yet distributed over the channels that went through the quantized layers. Then, the DenseToRes layer performs convolution with residual connections; the convolution operation distributes the preserved information to all channels, while the residual connections minimize the information loss due to quantized convolution operations by adding the undamaged input features to the output features [16]. As a result, this architecture allows for preserving

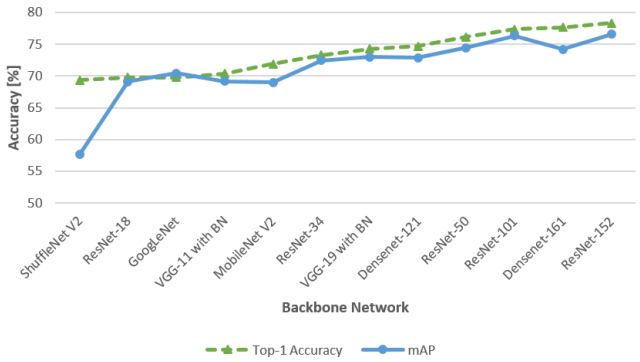

Fig. 1. Correlation between the object detection performance and classification accuracy of the backbone network.

TABLE I
PROPOSED BACKBONE NETWORK

Stage	Layer	
Stage 0	Initial layer	3×3 Conv w/ Stride 2
		2×2 Max Pool w/ Stride 2
		3×3 Depthwise Conv
		1×1 Conv
Stage 1	DenseToRes layer ×1	3×3 (1, 1)-bit Conv w/ Dense
		1×1 (1, 2)-bit Conv w/ Residual
	Transition layer	3×3 Depthwise Conv
		1×1 (1, 2)-bit Conv w/ Residual
		2×2 Max Pool w/ Stride 2
Stage 2	DenseToRes layer ×2	3×3 (1, 1)-bit Conv w/ Dense
		1×1 (1, 2)-bit Conv w/ Residual
	Transition	3×3 Depthwise Conv
		1×1 (1, 2)-bit Conv w/ Residual
		2×2 Max Pool w/ Stride 2
Stage 3	DenseToRes layer ×4	3×3 (1, 1)-bit Conv w/ Dense
		1×1 (1, 2)-bit Conv w/ Residual
	Transition layer	3×3 Depthwise Conv
		1×1 (1, 2)-bit Conv w/ Residual
		2×2 Max Pool w/ Stride 2
Stage 4	DenseToRes layer ×3	3×3 (1, 1)-bit Conv w/ Dense
		1×1 (1, 2)-bit Conv w/ Residual
	Transition layer	3×3 Depthwise Conv
		1×1 (1, 2)-bit Conv w/ Residual
Classification layer		7×7 Global Avg Pool
		1000-D Fully-Connected

∗ Bitwidth of conv : (Weight, Activation)

original information through dense connections and then immediately distributes this information to other channels through the residual connections that follow. We construct a backbone network that alternates between a DenseToRes layer and a transition layer, as detailed in Fig. 2 and Table I. The transition layer is used to reduce the horizontal and vertical size of the feature map. The initial layer also reduces the initial input size, reducing the computational cost of the following layers. These layers are optimized with depthwise convolution and quantization to increase computational efficiency. Below are more implementation details of the backbone network.

1) Layer Structure: We use the post-activation structure ordered as binarization–convolution–batch normalization–activation and RPReLU activation function. RPReLU was first proposed in [18], and it shifts the origin point of the PReLU function with two trainable channelwise biases.

2) DenseToRes Layer: In DenseToRes layers, we first use 3 × 3 full binary convolutions and concatenate input and output features for dense connections. Then, we use 1 × 1 pointwise binary-weight convolutions with residual connections. In this convolution, we use 2-bit input activations to minimize performance degradation.

3) Transition Layer: The structure of the transition layer is depicted in Fig. 3(b). In transition layers, we make the output channel size the same as the input channel size to improve performance, as suggested in [39], which is contrary to prior work on binarized DenseNet [19]. In experiments, we observe a large accuracy drop when transition layers are deeply quantized. Hence, the transition layers are kept in full-precision during training for the image classification task. They are later quantized into 8 bits during training for the object detection task, as described in Section III-C. We employ 3 × 3 depthwise convolutions as they require fewer parameters when the channel size increases in later layers compared to 1 × 1 pointwise convolutions. They are followed by residual connections identical to those in the DenseToRes layer.

4) Initial Layer: Fig. 3(c) displays the initial layer in our model. The initial layer reduces the input image size by 16× using stride and max pooling before the first dense layer, which significantly reduces the computational complexity of the following layers.

5) Training Scheme: As BNNs have a limited capacity to express features, it is hard to learn appropriate activation distributions with conventional training schemes. To address this issue, recent studies on BNNs [17], [18] propose to use a full-precision baseline as a target of training rather than directly using the ground truth. Then, the binarized model starts to produce similar output distribution to a full-precision baseline, which enhances accuracy. This transfer learning is also adopted for training our model. We use ResNet-34 as a teacher network and distributional loss with the output value of this network as in [17]. In addition, we use a two-step training scheme; we first train a network with binary activations and full-precision weights and then use the resulting model as the initial point to train a network with both binary weights and activations following the scheme in [17] and [18]. We use the RAdam optimizer [40] and the cosine annealing learning rate scheduler [41], as in [42].

Table II displays the ablation study results, which show the effects of each design optimization on the performance and model size of the backbone network. In this experiment, the model was trained on ImageNet for 120 epochs, and two-step learning was not used to reduce training overhead. Note that all non-binarized layers in the initial and transition layers are implemented with full precision. First, we build a baseline model using full binary convolution with dense connection and 1 × 1 full-precision convolution in transition layers. The model exhibits good accuracy but the model size is large due to 1 × 1 high-precision convolution in the transition layer. Starting from the baseline model, we try to reduce the model size while maintaining performance by applying each design optimization. Binarizing the transition layer greatly reduces the model size, but it exhibits a large performance degradation. To mitigate this performance degradation, we add a residual connection after the dense connection and achieve an improvement of 4.26% in accuracy. To improve accuracy further, high-precision depthwise convolution is added in the transition layer. As depthwise convolution has a small number

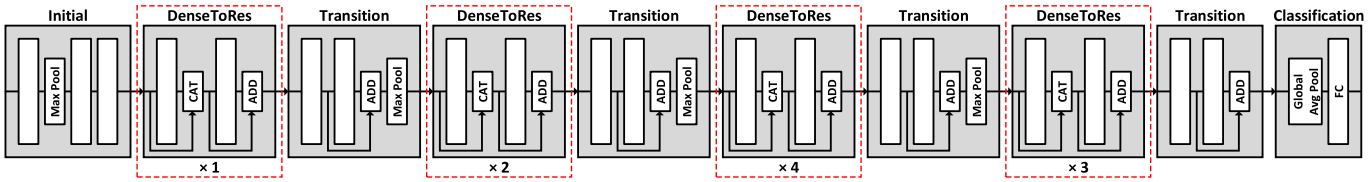

Fig. 2. Architecture of the proposed backbone model.

TABLE II

EFFECTS OF EACH DESIGN OPTIMIZATION ON PERFORMANCE AND MODEL SIZE OF BACKBONE NETWORK SIZE

Design optimization point		Progress to proposed backbone network				
DenseToRes layer	3×3 (1, 1)-bit conv w/ dense	✓	✓			
	3×3 (1, 1)-bit conv w/ dense + 1×1 (1, 1)-bit conv w/ residual			✓	✓	
	3×3 (1, 1)-bit conv w/ dense + 1×1 (1, 2)-bit conv w/ residual					✓
Transition layer	1×1 conv	✓				
	1×1 (1, 1)-bit conv w/ residual		✓	✓		
	3×3 depthwise conv + 1×1 (1, 1)-bit conv w/ residual				✓	
	3×3 depthwise conv + 1×1 (1, 2)-bit conv w/ residual					✓
Top-1 Accuracy		63.89	53.68	57.94	61.52	64.51
Model Size (MB, w/o FC layer)		3.50	0.40	0.71	0.79	0.79

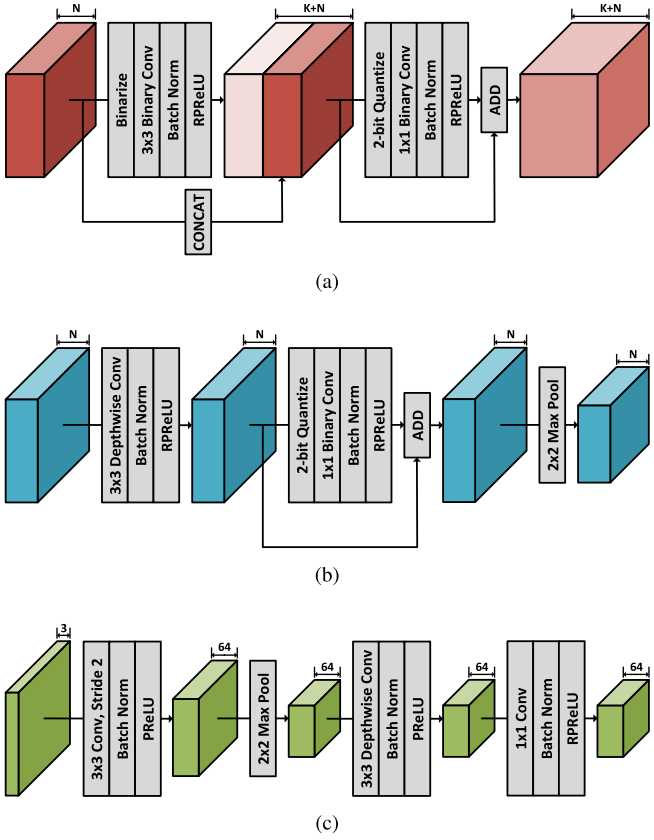

Fig. 3. (a) DenseToRes layer, (b) transition layer, and (c) initial layer of the proposed model.

of parameters compared to conventional convolution, the increase in model size is relatively small even if implemented in high precision. Through this optimization, performance is increased by 3.58%, while increasing the model size only by 0.08 MB. Finally, by increasing the bitwidth of input activation in the convolution with residual connection to 2 bits, the proposed model outperforms the baseline. This optimization does not increase the model size since only the bitwidth of activation is increased. Through the optimization techniques discussed above, our proposed backbone network achieves good performance with a significantly smaller model size.

TABLE III

COMPARISONS OF DIFFERENT QUANTIZATION SCHEMES IN THE BACKBONE NETWORK

	2-bit	4-bit	6-bit	Proposed
Top-1 Accuracy (%)	50.49	63.30	65.68	62.98
Model Size (MB, w/o FC layer)	1.21	2.39	3.57	0.64

One may consider employing uniform quantization (i.e., quantizing all activations and weights into identical precision) instead of the mixed-precision quantization we proposed. For comparisons, we conducted additional experiments and observed the performance when representing those values in an intermediate uniform precision (Table III). Each model was trained on ImageNet for 120 epochs, and two-step learning was not used to reduce training overhead. Using 6- and 4-bit quantization improves the accuracy by 2.7% and 0.32%, respectively, but the model size is significantly increased by 5.6× and 3.7×. On the other hand, 2-bit quantization results in a large performance drop of 12.49% while still having a 1.75× larger model size. Therefore, the proposed model is an optimal design point in the accuracy-model size trade-off.

B. Head Network Design and Detection Scheme

The head network is designed with a structure similar to that of the backbone network. The last classification layer is removed from the pretrained backbone network, and five dense layers and two transition layers are attached at the end as a head network. In the transition layer of the head network, max pooling is not performed. In the last layer, 1 × 1 high-precision convolution is used to perform detection. The entire network is trained for the object detection task on PASCAL VOC 2007 + 2012 dataset [43].

The network predicts five bounding boxes that possibly represent one of 20 types of objects in each grid, which is similar to the detection scheme of YOLOv2. When training the model for the object detection task, YOLOv2 uses the mean squared error (MSE) loss as a loss function. On the other hand, we use binary cross entropy (BCE) loss for class

and confidence score to improve training performance as in YOLOv3 [7].

C. Layer Fusion and Quantization

While most layers in the proposed object detection model are binarized, the initial layer, transition layers, and the last detection layer are implemented in higher precision as the object detection accuracy is largely affected by computation errors in those layers. We quantize the activations and weights of those layers into 8 bits with uniform nonzero quantization (1) to minimize memory footprint and computational complexity. We also employ scaling factors in quantization, but their values are constrained to powers of 2 so that quantization could be implemented using simple shift operations. In those 8-bit layers, we fuse the batch normalization layer into the convolution layer to minimize hardware overheads. In other binarized layers such as DenseToRes layers, the batch normalization layer, and RPReLU are not fused and are quantized into 8 bits. The detailed structure of our object detection model is presented in Table IV.

D. Training Result of Proposed Object Detection Model

First, we implement the proposed object detection model in software using the Pytorch framework [44] on Nvidia RTX 2080 Ti GPUs. The backbone network is trained on the ImageNet dataset with 512 batch size on four GPUs. The initial learning rate is set to 2E-3 and the training continues for 120 epochs in both steps of the two-step training described in Section III-A. We use 1E-5 weight decay factor in training step 1 and set it to 0 in step 2 as used in [17] and [18]. After two-step training, the backbone network exhibits 65.54% image classification accuracy on the ImageNet validation set. Then, the entire model including the head network is trained for the object detection task on the PASCAL VOC 2007 + 2012 dataset with 16 batch size. The initial learning rate is set to 1E-4. We also construct custom quantization functions in PyTorch, so that the software model exactly matches the hardware. The resulting model exhibits mAP score of 64.92% when tested on the PASCAL VOC 2007 test set. The model size is 1.80 MB.

TABLE IV
NETWORK STRUCTURE OF OUR OBJECT DETECTION MODEL

Network	Stage	Layer[a]	Kernel / Stride	Bit-width[b]	Output Size
Input					416×416×3
Backbone Network	Stage 0	C	3×3 / 2	(8, 8)	104×104×64
		M	2×2 / 2		
		DC	3×3 / 1	(8, 8)	
		C	1×1 / 1	(8, 8)	
	Stage 1	DTR×1	3×3 / 1	(1, 1)	52×52×128
			1×1 / 1	(1, 2)	
		T	3×3 / 1	(8, 8)	
			1×1 / 1	(1, 2)	
		M	2×2 / 2		
	Stage 2	DTR×2	3×3 / 1	(1, 1)	26×26×256
			1×1 / 1	(1, 2)	
		T	3×3 / 1	(8, 8)	
			1×1 / 1	(1, 2)	
		M	2×2 / 2		
	Stage 3	DTR×4	3×3 / 1	(1, 1)	13×13×512
			1×1 / 1	(1, 2)	
		T	3×3 / 1	(8, 8)	
			1×1 / 1	(1, 2)	
		M	2×2 / 2		
	Stage 4	DTR×3	3×3 / 1	(1, 1)	13×13×704
			1×1 / 1	(1, 2)	
		T	3×3 / 1	(8, 8)	
			1×1 / 1	(1, 2)	
Head Network	Stage 5	DTR×3	3×3 / 1	(1, 1)	13×13×896
			1×1 / 1	(1, 2)	
		T	3×3 / 1	(8, 8)	
			1×1 / 1	(1, 2)	
	Stage 6	DTR×2	3×3 / 1	(1, 1)	13×13×1024
			1×1 / 1	(1, 2)	
		T	3×3 / 1	(8, 8)	
			1×1 / 1	(1, 2)	
	Detection	C	1×1 / 1	(8, 8)	13×13×125

(a) C: Convolution, M: Max Pool, DC: Depthwise Convolution, DTR: DenseToRes layer, T: Transition layer
(b) Bitwidth: (Weight, Activation)

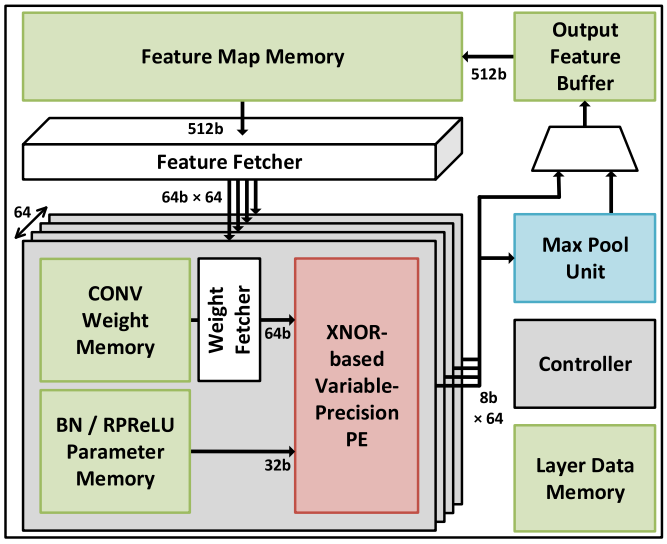

Fig. 4. Overview of the proposed object detection processor.

IV. REAL-TIME OBJECT DETECTION PROCESSOR

A. Processor Architecture

Fig. 4 displays the overall architecture of the proposed object detection processor. The processor has a unified feature map memory implemented with dual-port SRAM. The feature map memory reads 64 8-bit data at a time and transmits the data to PEs depending on the operation of each layer. The processor has 64 PEs in total, and hence 64 outputs are obtained in parallel. Each PE has two memory blocks that store the weights of the convolution layer and the parameters of batch normalization and RPReLU. We optimize the architecture to store all the network parameters in ON-chip memory since external memory access is very costly and often a processing bottleneck in deep learning accelerators. Specifically, we choose the output-stationary scheme depicted in Fig. 5. While other processing schemes such as weight-stationary allow for reusing weights, they necessitate a larger buffer to temporarily store intermediate results. Contrarily, our processor only needs to store a single output pixel at a time, significantly reducing hardware resources and securing enough ON-chip memory space to store network parameters.

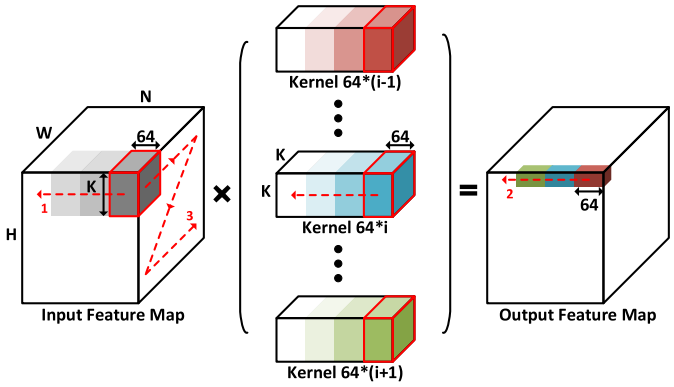

Fig. 5. Example of convolution layer processing.

TABLE V
COMPARISONS OF PROCESSING SCHEMES

	Weight Stationary	Row Stationary	Output Stationary
Input buffer size	$N\times W\times H\times B_i$	$N\times K\times W\times B_i$	$N\times K\times K\times B_i$
Output buffer size	$C_o\times W\times H\times B_o$	$C_o\times W\times B_o$	$C_o\times B_o$
Weight read times	1	H	W×H

Table V summarizes the hardware overhead of three different processing schemes, where B_i and B_o are the bitwidth of input activation and output accumulation, respectively, and C_o is the number of output channels calculated simultaneously in the processor. H, W, and N represent the dimensions of input feature map, and K is the filter size as depicted in Fig. 5. In the weight-stationary scheme, each weight is reused for the entire input feature map. The input feature map is processed by each of C_o weight blocks. To produce the final C_o output feature maps, temporary output accumulations with the same size and the entire input feature maps need to be stored, necessitating large ON-chip buffers. For the row-stationary scheme, the size of input and output buffers is smaller as only some rows of input features are processed and stored at a time, but it still requires larger buffers than the output-stationary scheme. With our configuration of hardware parallelism, the buffer size is 1138× and 9× larger for weight-stationary and row-stationary schemes, respectively. This is especially critical in resource-constrained platforms such as FPGA devices. Due to output stationary processing, we need to load a new weight value for each input feature. However, they are read from internal memory, and the energy savings due to removing external DRAM access exceeds this overhead. In the proposed DenseToRes-Net model, the number of channels increases by 64 after each DenseToRes layer. Hence, we can easily split the output channels into blocks of 64, where each block is processed by 64 PEs at a time. In other words, each PE is responsible for generating one output channel.

B. Variable-Precision PE

In most BNN models, not all layers are binarized to suppress performance degradation; for instance, the first layer and the last classification layer are susceptible to performance drop due to deep quantization, so they still remain in high precision [15], [16], [17], [18], [19], [25], [42]. However, this necessitates an additional arithmetic unit to support high-precision operations. To mitigate this issue, we introduce a unified PE that supports both 1-bit and multibit MACs using simple bitwise operations through optimizing quantization scheme.

Assuming that 1-bit quantized activation $x \in \{-1, 1\}$ is represented by a binary value $\hat{x} \in \{0, 1\}$, multiplications with 1-bit weights $w \in \{-1, 1\}$ are realized using XNOR operations as below

$$x = 2\hat{x} - 1 \tag{6}$$

$$w = 2\hat{w} - 1 \tag{7}$$

$$x \cdot w = 2 \cdot (\hat{x} \odot \hat{w}) - 1 \tag{8}$$

where $\odot$ represents bitwise XNOR operation. Then, the accumulation of multiplication results is substituted by a simple bit count operation. If we extend the transformation in (6) and (7) to n bits, an n-bit binary number $\hat{X}$ in the circuit translates to an integer X which is represented in n-digit binary number format with signed digits (i.e., $X = (x_{n-1} \ldots x_1 x_0)_2$ where $x_k \in \{-1, 1\}$, and $\hat{X} = (\hat{x}_{n-1} \ldots \hat{x}_1 \hat{x}_0)_2$ where $\hat{x}_k \in \{0, 1\}$)

$$\begin{aligned} X &= (x_{n-1} \ldots x_1 x_0)_2 \\ &= x_{n-1} \cdot 2^{n-1} + \cdots + x_1 \cdot 2^1 + x_0 \\ &= (2\hat{x}_{n-1} - 1) \cdot 2^{n-1} + \cdots + (2\hat{x}_0 - 1) \\ &= 2 \cdot (\hat{x}_{n-1} 2^{n-1} + \cdots + \hat{x}_0) - (2^{n-1} + \cdots + 1) \\ &= 2\hat{X} - (2^n - 1) = 2 \cdot (\hat{X} - 2^{n-1}) + 1. \end{aligned} \tag{9}$$

Note that the circuit can represent only odd values since $X \in \{-2^n + 1, -2^n + 3, \ldots, 2^n - 1\}$ for $\hat{X} \in \{0, \ldots, 2^n - 1\}$. Consider multiplication of activation X and weight W which are represented using n and m digits in the binary number format with signed digits of $\{-1, 1\}$. According to (6)–(8), multiplication can be implemented as

$$\begin{aligned} X \cdot W &= (x_{n-1} \cdot 2^{n-1} + \cdots + x_0) \cdot (w_{m-1} \cdot 2^{m-1} + \cdots + w_0) \\ &= (x_{n-1} \cdot w_{m-1}) \cdot 2^{m+n-2} + \cdots + (x_0 \cdot w_0) \\ &= (2 \cdot (\hat{x}_{n-1} \odot \hat{w}_{m-1}) - 1) \cdot 2^{m+n-2} + \cdots \\ &\quad + (2 \cdot (\hat{x}_0 \odot \hat{w}_0) - 1) \\ &= 2 \cdot \{(\hat{x}_{n-1} \odot \hat{w}_{m-1}) \cdot 2^{m+n-2} + \cdots + (\hat{x}_0 \odot \hat{w}_0)\} \\ &\quad - (2^{m+n-2} + \cdots + 1) \\ &= 2 \cdot \{(\hat{x}_{n-1} \odot \hat{w}_{m-1}) \cdot 2^{m+n-2} + \cdots + (\hat{x}_0 \odot \hat{w}_0)\} \\ &\quad - (2^{n-1} + \cdots + 1) \cdot (2^{m-1} + \cdots + 1) \\ &= 2 \cdot (\hat{X} \odot \hat{W}) - (2^{n-1} + \cdots + 1) \cdot (2^{m-1} + \cdots + 1). \end{aligned} \tag{10}$$

This suggests that multibit multiplication $X \cdot W$ can be implemented using simple bitwise XNOR operations along with a linear transform of $y = 2x - b$. Fig. 6 displays an example of multiplying $+5$ $(= (+1) \cdot 2^2 + (+1) \cdot 2^1 + (-1) \cdot 2^0)$ by -3 $(= (-1) \cdot 2^1 + (-1) \cdot 2^0)$. It should be noted that this only applies when multiplying odd values since X and W can only have odd values as discussed above. Hence, we employ uniform nonzero quantization as the quantization scheme in our hardware since

$$
\begin{array}{lcccc}
 & & +1 & +1 & -1 \\
\times & & & -1 & -1 \\
\hline
 & & -1\cdot+1 & -1\cdot+1 & -1\cdot-1 \\
+ & -1\cdot+1 & -1\cdot+1 & -1\cdot-1 & \\
\hline
 & & 2\cdot(0\odot1)-1 & 2\cdot(0\odot1)-1 & 2\cdot(0\odot0)-1 \\
+ & 2\cdot(0\odot1)-1 & 2\cdot(0\odot1)-1 & 2\cdot(0\odot0)-1 & \\
\hline
2\cdot\Big(& & 0\odot1 & 0\odot1 & 0\odot0\Big) \\
 & 0\odot1 & 0\odot1 & 0\odot0 & \\
+ & & & & -(2^2+2^1+2^0)(2^1+2^0) \\
\hline
 & & 2\cdot((1\cdot2^0)\cdot2^1+(1\cdot2^0)\cdot2^0)-(2^2+2^1+2^0)(2^1+2^0) & &
\end{array}
$$

Fig. 6. Example of XNOR-based multiplication of 3- and 2-bit binary numbers with signed digits.

it can be configured to produce odd numbers only. More specifically, (1) is reformulated as

$$
\begin{aligned}
x_q &= s \times \left(\text{round}\left(\frac{x_r}{s}+0.5\right)-0.5\right) \\
&= s \times \left(\text{floor}\left(\frac{x_r}{s}\right)+0.5\right) \\
&= \frac{s}{2} \times \left(2\text{floor}\left(\frac{x_r}{s}\right)+1\right). \qquad (11)
\end{aligned}
$$

Then, we can store $\text{floor}(x_r/s) + 2^{n-1}$ in the circuit following (9), and this is achieved by simply inverting MSB. Therefore, if we use uniform nonzero quantization for convolution, multibit multiplications can be realized using XNOR operations.

Utilizing the property discussed above, we propose a variable-precision MAC unit shown in Fig. 7(a). The unit supports high-precision MAC operations through bit slicing. An operation is first decomposed into bitwise operations, and then the results are combined using shift-add operations to produce the desired results. The MAC unit is capable of processing 64 bitwise XNOR operations in parallel, and their results are combined differently in the shift-adder tree depending on the required precision. Specifically, the adders in stage 1 are configured by the number of activation bits. In stages, 2 through 4, the configurations of the shift-adders are determined by the number of weights bits. With 64 XNOR gates, the MAC unit can implement multiple high-precision MACs in parallel. Some possible configurations are shown below:

1) 64 1 b × 1 b MACs;
2) 32 2 b × 1 b or 16 2 b × 2 b MACs;
3) 16 4 b × 1 b, eight 4 b × 2 b, or four 4 b × 4 b MACs;
4) eight 8 b × 1 b, four 8 b × 2 b, two 8 b × 4 b, or one 8 b × 8 b MACs.

The input bits should be rearranged to support MAC operations in different precision, which would incur hardware overhead due to additional multiplexers. Therefore, we only implement the input path for 1 b × 1 b, 2 b × 1 b, and 8 b × 8 b MACs, which are required in the proposed object detection algorithm. Finally, the MAC operation is completed by performing the linear transform in (10) on the accumulation result through the shift-add operation. Note that the bias value should be multiplied by the number of operands, and it is precomputed and stored in hardware for each layer in the network.

Fig. 8 displays the PE in our object detection processor employing the variable-precision MAC unit. After the variable-precision MAC unit, we employ a conventional MAC unit with 16 b × 8 b precision to implement batch normalization. The RPReLU activation function multiplies the input with a weight when the input is negative, which is implemented using a MUX that takes the sign bit of the MAC result as a control signal. Also, the result should be shifted according to the scale of the RPReLU weight. After adding the bias of RPReLU, the result is quantized to 8 bits using the output scale. Since all quantization scales are in the form of a power-of-2, all scale-related operations can be implemented using simple shift operations.

C. Memory and Datapath

The feature map memory stores 21 632 512-bit words. To support 8-bit convolutions and residual connections, all activations are stored as 8-bit values. Each row in the feature map memory stores 8-bit features from 64 channels. The feature fetcher, connected to the feature map memory, reads 64 8-bit values from memory and sends them to each PE according to the bit configuration of each layer. For 1 b × 1 b operation, only the sign bits of 64 values are selected and shared across PEs. For 2 b × 1 b operation, 32 out of 64 values are quantized into 2 bits and shared across PEs, while the remaining 32 values are distributed in the next cycle. Thus, it consumes a 512-bit word in two cycles. For 8 b × 8 b operation, one 8-bit value is sent to all PEs, and hence a 512-bit word is used for 64 cycles. However, in 8 b × 8 b depthwise convolution, it has a different datapath because there is no operation between input channels. About 64 data are sent to 64 different PEs, consuming 512-bit data in one cycle. The convolution weight memory in each PE sends weight values to the MAC unit according to the bit configuration of each layer through a weight fetcher, whose operation is similar to the feature fetcher. About 512-bit data produced by PE or the max pooling module are first stored in the output feature buffer. Since our processor architecture employs a unified feature map memory, the address to which the output feature is to be written may not be available if the input feature in that address has not been processed yet. Therefore, we temporarily store output features in the buffer until the destination address becomes available. Our processor reads layer configurations from the layer data memory before processing each layer. They include feature map size, kernel size, and type of layer, and the processor uses this information to check when an input feature at a specific address is no longer used. In the convolution operation, the number of output features that require a specific input feature is determined by the kernel size and channel size. Therefore, if all the output features related to an input feature are obtained, this input feature is no longer used. As our processor employs output-stationary processing, it calculates output features in the order depicted in Fig. 5. Therefore, the processor temporarily stores calculated output features in the output feature buffer and then overwrites them in the addresses of the input features that are no longer needed. This logic is

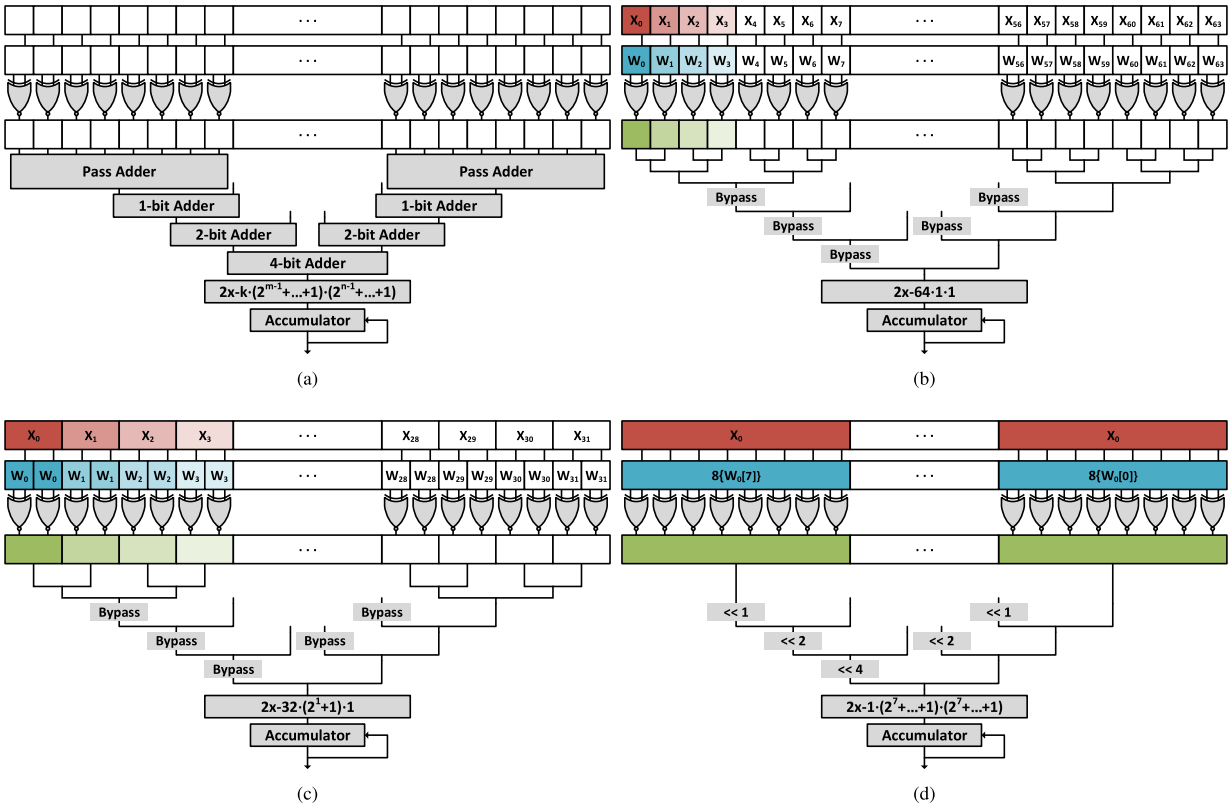

Fig. 7. (a) Architecture of the proposed variable-precision MAC unit and configurations for (b) 64 1 b × 1 b MACs, (c) 32 2 b × 1 b MACs, and (d) 1 8 b × 8 b MAC.

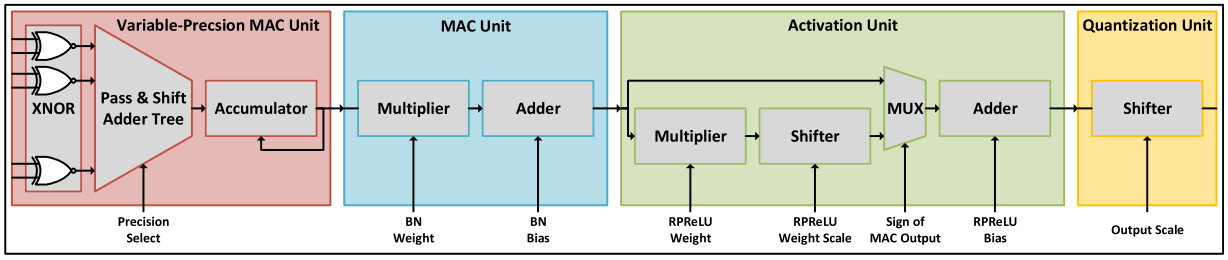

Fig. 8. Architecture of XNOR-based variable-precision PE.

implemented in the controller, which informs the memory and the buffer when to write the output.

V. Experimental Results

A. Demonstration System

To validate the proposed object detection model and processor, we implement the design on the Xilinx ZCU102 evaluation board. The board features Xilinx Zynq UltraScale+ MPSoC device [46] which consists of a general-purpose CPU (*Processing System*) and reconfigurable FPGA logic (*Programmable Logic*), providing a high degree of flexibility. Fig. 9 shows the overall architecture of the demonstration system.

The system realizes end-to-end processing of object detection from capturing images to displaying detection results overlayed on the input image (Fig. 11). The proposed object detection processor is implemented in FPGA logic, and it runs the object detection model described in Section III. Data transfer between CPU and FPGA are handled by Xilinx interface IPs using AXI protocol [47]. All model parameters are stored in the processor, and the data channel for input images is implemented using a 512-bit AXI-Stream interface along with a control channel using a 32-bit AXI-Lite interface. The CPU and the object detection processor exchange START and DONE signals through the AXI-Lite interface, while the input image and the network output go through the AXI-Stream interface. The entire process is as follows. When the CPU opens the saved image file or a video stream from the USB port, it is resized to 416 × 416 and normalized to match the input resolution of the model. Resized images are copied to the shared memory area of DRAM. The DMA IP on the FPGA transfers the data to the object detection processor and writes the object detection results back to DRAM. The CPU converts the data type of the object detection results from integer to 32-bit floating point, reorders the data sequence to match the processing order of the CPU, and copies the 8-bit data between different memory spaces using NEON [48]. After the CPU receives the results from FPGA, it performs NMS and overlays the location and object type of the detected objects on the image, which is displayed via output channels such as Ethernet and DisplayPort.

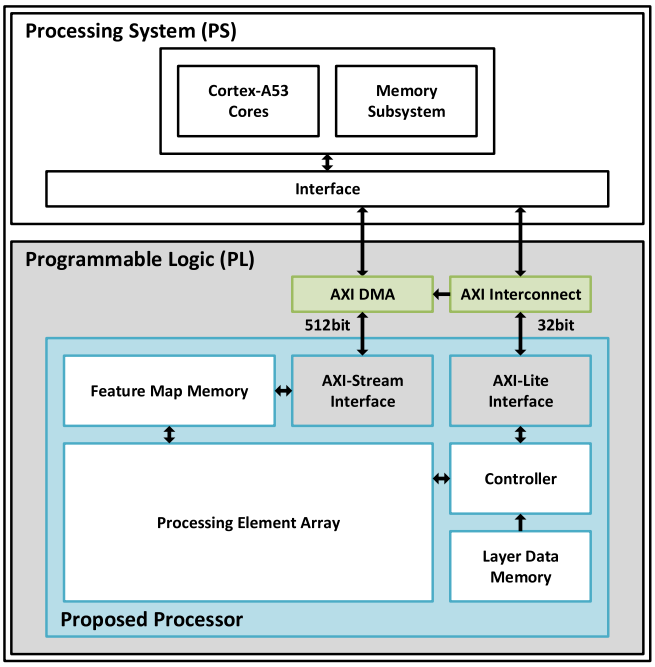

Fig. 9. Demonstration system overview (modified from [45]).

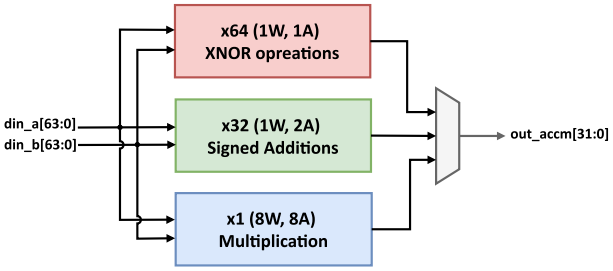

Fig. 10. Schematic of baseline MAC unit for comparison.

TABLE VI
HARDWARE OVERHEAD COMPARISON

Resource	Baseline	Proposed PE	Reduction
LUT	558	440	21.15%
Flip-Flop	78	62	20.51%
CARRY8	30	18	40.00%

B. Implementation Results

Our object detection processor utilizes variable-precision MAC units described in Section IV-B. We compare the implementation costs of our design and a baseline that has independent datapaths dedicated for (1 W, 1 A), (1 W, 2 A), and (8 W, 8 A) MAC operations. Fig. 10 shows the schematic of the baseline that exhibits the same throughput as our design. Table VI displays the hardware implementation costs of those two designs when synthesized for the target FPGA device. Experimental results show that the proposed variable-precision MAC unit reduces the number of LUTs, flip-flops, and carry logics (CARRY8) by 21.15%, 20.51%, and 40.00%, respectively.

Our object detection processor design is synthesized and place-and-routed in Xilinx Vivado 2018.3. The target frequency is set to 300 MHz. Hardware resource utilization is summarized in Table VII. The variable-precision MAC unit only uses XNOR gates and adder trees, and those blocks are synthesized into only LUTs and flip-flops [49]. Consequently, DSP slices [50] are not used for convolution operations, and they are only used to implement batch normalization, RPReLU, and the control unit generating read/write addresses for ON-chip memory. All the blocks except BRAMs are utilized under 50%, whereas BRAMs show a higher utilization of 84.76% since we store all the model parameters and activations in the processor to avoid costly external DRAM access. In our object detection model, the largest feature map has a size of 10.563 Mb. Considering the granularity of BRAM, 10.775 Mb space is allocated to the activations, which is equivalent to 33.6% BRAM utilization. The size of the model parameters is 14.37 Mb, and 15.750 Mb BRAM space is allocated to the parameters, which translates to 49.1% BRAM utilization.

TABLE VII
HARDWARE RESOURCE UTILIZATION

Resource	Utilization	Available	Utilization(%)
LUT	110,814	274,080	40.43
LUTRAM	6,515	144,000	4.52
Flip-Flop	74,533	548,160	13.60
BRAM (36Kbit)	773	912	84.76
DSP	132	2,520	5.24

C. Performance Evaluations

We measure the processing latency of the processor as the time difference between when the START command is issued by the CPU and when the CPU receives the DONE signal from the processor. When the processor runs at 300 MHz clock frequency, the processing latency is measured at 15.5 ms, which translates to a throughput of 64.51 frames/s.

Table VIII compares our design against prior FPGA-based object detection processors. Since they use different bit precision, we calculate the BitGOPs [51], [52], [53] for each design, which represents the operation count normalized with respect to the bit precision considering actual hardware implementation costs. In general, our object detection model achieves a competitive object detection accuracy, while the number of parameters and operations are lower than those of other models with a similar object detection accuracy. The designs in [29] and [31] achieve low BitGOPs and small model sizes, but their accuracy and frame rate are significantly lower than our design. The designs in [28] and [30] exhibit similar mAP to ours, but their models have a larger size and require significantly more operations in BitGOPs, which translate to more hardware resources such as DSP, LUT, and FF. In addition, the design in [28] uses a costly full-precision SVR as a detector, resulting in a lower frame rate. The design in [32] achieves a higher mAP score since the model is not deeply quantized and it is trained on a larger train set including the MS COCO dataset [54]. This design also exhibits a high frame rate of 84.8 frames/s, but using high-precision weights and activations unavoidably requires a large amount of external memory access and more complicated computing units.

For comparison between designs, we first calculate the implementation cost as the number of equivalent LUTs. Each FPGA device has different types of DSP and LUT, but we can

TABLE VIII
COMPARISONS WITH PRIOR FPGA-BASED OBJECT DETECTION PROCESSORS

	Tincy YOLO [29]	Lightweight YOLO-v2 [28]	SIM-YOLO-v2 [30]	CoDeNet [31]	Real-Time SSDLite [32]	This Work
Platform	ZU+ XCZU3EG (16nm)	ZU+ XCZU9EG (16nm)	Virtex7 XC7VX485T (28nm)	ZU+ XCZU3EG (16nm)	Arria10 SoC 10 (20nm)	ZU+ XCZU9EG (16nm)
Train Set	VOC 07+12	VOC 07+12	VOC 07+12	VOC 07+12	VOC 07+12 +COCO	VOC 07+12
Test Set	VOC 07	VOC 07	VOC 07	VOC 07	VOC 07	VOC 07
mAP	48.5	67.6	64.16	51.1	78.6	64.92
GOPs	4.5	14.97	17.18	0.58	1.45	9.95
BitGOPs[a]	16.88	303.00	99.00	18.56	363.73	40.84
Model Size (MB)	0.82	4.76	1.88	0.76	6.69	1.80
External DRAM Bandwidth (MB/s)	N/A	194.26	0	203.37	3663.46	0
Precision (W,A)[b]	(1,3)	(1,1) (32,32)	(1,4-6)	(4,8)	(16,16)	(1,1), (1,2), (8,8)
Image Size	416×416	224×224	416×416	256×256	320×320	416×416
Frame Rate (FPS)	16	40.81	60.72[c]	32.20	84.80	64.51
Power (W)	N/A	4.50	11.11[c]	5.60	9.80	6.58
Frequency (MHz)	N/A	300	200	250	200	300
BRAM (MB)	N/A	3.75	2.51	0.95	0.97	3.40
DSPs	N/A	377	272	360	980	132
LUTs	N/A	135k	155k	34k	94k[d]	117k
FFs	N/A	370k	115k	42k		74k
Equivalent LUTs	N/A	330k	286k	220k	807k	185k
Power efficiency (FPS/W)	N/A	9.07	5.47	5.75	8.65	9.80
mAP×FPS/kLUT[e]	N/A	8.36	13.62	7.48	8.26	22.64

(a) Excluding batch normalization as its precision is not reported in other designs.
(b) W: Weight, A:Activation
(c) Without batch processsing
(d) ALMs in Intel FPGA
(e) Equivalent LUTs of computing units

directly compare the hardware cost by counting the number of LUTs to realize the same function. For Xilinx FPGAs, a DSP slice in 7-Series devices (28 nm) can be configured as a 25 b × 18 b 2's-complement multiplier [55], whereas a DSP slice in UltraScale+ devices (16 nm) can be configured as a 27 b × 18 b 2's-complement multiplier [50]. Those multipliers are synthesized into 483 and 517 LUTs, respectively. For Intel FPGAs, a DSP slice can be configured as two 18 b × 18 b 2's-complement multipliers [32], and they are synthesized into 728 LUTs. We also conservatively convert an ALM in Intel FPGAs into one LUT and one FF.

However, it is not fair to simply compare the number of equivalent LUTs, since each design exhibits different throughput (frame rate) and accuracy (mAP). Therefore, here we define a new figure-of-merit (FoM) that can reflect both accuracy and power consumption

$$\text{FoM} = \frac{\text{mAP} \times \text{Frame Rate}}{\text{Number of Equivalent LUTs}}. \tag{12}$$

This metric is based on the assumption that the power consumption is proportional to the number of LUTs in the design in a naïve implementation, and we are interested in the energy consumption per frame (i.e., power consumption divided by frame rate). Also, higher accuracy (mAP) tends to incur larger power consumption as the model size and complexity increases. Table VIII confirms that our design exhibits 66% higher FoM than the prior art. Note that this analysis does not include power consumption due to external memory access, and hence the advantage of our design will become even more significant when its effect is considered.

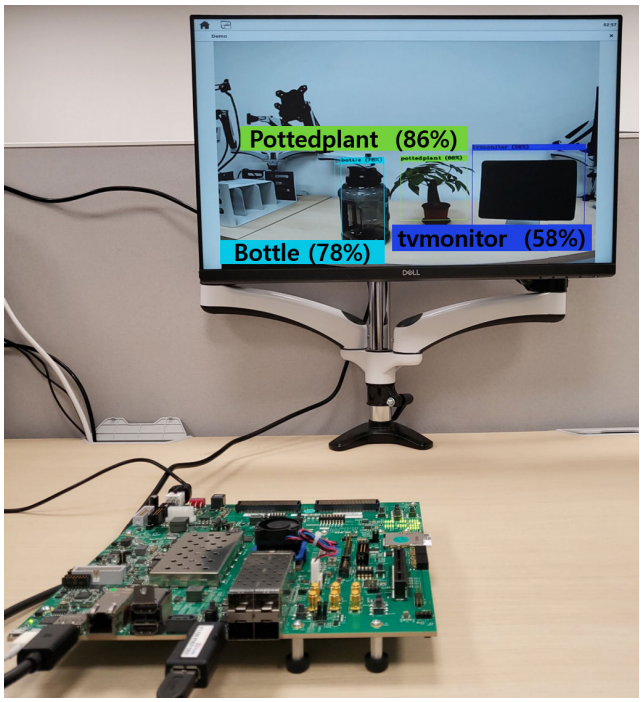

Fig. 11. Test setup for end-to-end processing demonstration.

The power consumption of the device could be measured using Xilinx Power Advantage Tool [56]. The tool reports the voltage and current of each power rail measured by the onboard power monitoring and management ICs [57]. Table IX shows the power consumption of three power domains when the system is idle (i.e., only Linux OS is running) and

TABLE IX
POWER CONSUMPTION

Power Domain	Idle (mW)	Active (mW)
Full Power Domain	1343.6	1426.0
Programmable Logic Domain	2281.7	6582.0
TOTAL	3625.3	8008.0

active (i.e., object detection is running): full power domain (FPD) includes quad ARM Cortex-A53 cores, low-power domain (LPD) includes dual ARM Cortex-R5 cores, and programmable logic domain (PLD) includes FPGA. Since we do not use Cortex-R5 cores in the demonstration system, we excluded the power consumption of LPD. The clock frequency of FPGA and ARM Cortex-A53 is 300 MHz and 1.2 GHz, respectively.

VI. CONCLUSION

In this article, we propose a real-time object detection processor suitable for embedded platforms. We present a binarized object detection model, and it achieves a competitive accuracy with fewer parameters and operations through an optimal network structure employing the proposed DenseToRes layers. We also propose an efficient object detection processor that realizes real-time processing with limited hardware resources. The design utilizes resource-efficient variable-precision MAC units and completely removes the need for external DRAM access by optimizing the processing scheme. Implemented in FPGA, the object detection processor achieves 64.51 frames/s throughput with 64.92 mAP accuracy. Compared to prior FPGA-based designs for object detection, our design reduces hardware implementation costs as well as external memory bandwidth while maintaining high detection accuracy.

REFERENCES

[1] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 28, 2015, pp. 1–14.

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

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

[5] A. R. Pathak, M. Pandey, and S. Rautaray, "Application of deep learning for object detection," *Proc. Comput. Sci.*, vol. 132, pp. 1706–1717, Jan. 2018.

[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[7] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[8] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[9] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10781–10790.

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

[11] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[13] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4820–4828.

[14] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2016, pp. 525–542.

[16] 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. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 722–737.

[17] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–11. [Online]. Available: https://openreview.net/forum?id=BJg4NgBKvH

[18] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, "ReactNet: Towards precise binary neural network with generalized activation functions," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2020, pp. 143–159.

[19] J. Bethge, H. Yang, M. Bornstein, and C. Meinel, "BinaryDenseNet: Developing an architecture for binary neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 1951–1960.

[20] Z. Wang, Z. Wu, J. Lu, and J. Zhou, "BiDet: An efficient binarized object detector," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2049–2058.

[21] K. Ando et al., "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.

[22] F. Conti et al., "XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2940–2951, Nov. 2018.

[23] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.

[24] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "ChewBaccaNN: A flexible 223 TOPS/W BNN accelerator," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[25] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 29, 2016, pp. 1–9.

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

[27] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[28] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2018, pp. 31–40.

[29] T. B. Preuser, G. Gambardella, N. Fraser, and M. Blott, "Inference of quantized neural networks on heterogeneous all-programmable devices," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 833–838.

[30] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.

[31] Q. Huang et al., "CoDeNet: Efficient deployment of input-adaptive object detection on embedded FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2021, pp. 206–216.

[32] S. Kim, S. Na, B. Y. Kong, J. Choi, and I.-C. Park, "Real-time SSDLite object detection on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 6, pp. 1192–1205, Jun. 2021.

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[34] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7263–7271.
[35] X. Chen, J. Xu, and Z. Yu, "A 68-mW 2.2 Tops/W low bit width and multiplierless DCNN object detection processor for visually impaired people," *IEEE Trans. Circuits Syst. Video Techn. (CSVT)*, vol. 29, no. 11, pp. 3444–3453, Nov. 2019.
[36] S. Yin et al., "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 968–982, Dec. 2017.
[37] J.-S. Park et al., "9.5 A 6K-MAC feature-map-sparsity-aware neural processing unit in 5nm flagship mobile SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 152–154.
[38] A. Agrawal et al., "A 7nm 4-core AI chip with 25.6TFLOPS hybrid FP8 training, 102.4 TOPS INT4 inference and workload-aware throttling," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 144–146.
[39] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Red Hook, NY, USA: Curran Associates, 2018, pp. 1–77. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/9908279ebbf1f9b250ba689db6a0222b-Paper.pdf
[40] L. Liu et al., "On the variance of the adaptive learning rate and beyond," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–14. [Online]. Available: https://openreview.net/forum?id=rkgz2aEKDr
[41] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," 2016, *arXiv:1608.03983*.
[42] J. Bethge, C. Bartz, H. Yang, Y. Chen, and C. Meinel, "MeliusNet: An improved network architecture for binary neural networks," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2021, pp. 1439–1448.
[43] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Sep. 2009.
[44] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32. Vancouver, BS, Canada, Dec. 2019, pp. 8026–8037.
[45] L. Crockett, D. Northcote, C. Ramsay, F. Robinson, and R. Stewart, *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications*. Glasgow, Scotland: Strathclyde Academic Media, 2019.
[46] XILINX. (2023). *Zynq Ultrascale+ Device Technical Reference Manual*. Accessed: Jan. 28, 2023. [Online]. Available: https://docs.xilinx.com/r/en-U.S./ug1085-zynq-ultrascale-trm/Zynq-UltraScale-Device-Technical-Reference-Manual
[47] ARM. *Amba AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite Ace and Ace-Lite*. Accessed: Jan. 28, 2023. [Online]. Available: https://developer.arm.com/documentation/ihi0022/e
[48] ARM. *Fundamentals of ARMV8 Neon Technology*. Accessed: Jan. 28, 2023. [Online]. Available: https://developer.arm.com/documentation/102474/0100/Fundamentals-of-Armv8-Neon-technology
[49] XILINX. (2017). *Ultrascale Architecture Configurable Logic Block User Guide*. Accessed: Jan. 28, 2023. [Online]. Available: https://docs.xilinx.com/v/u/en-U.S./ug574-ultrascale-clb
[50] XILINX. *Ultrascale Architecture DSP Slice User Guide*. Accessed: Jan. 28, 2023. [Online]. Available: https://docs.xilinx.com/v/u/en-U.S./ug579-ultrascale-dsp
[51] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of ConvNets via differentiable neural architecture search," 2018, *arXiv:1812.00090*.
[52] Z. Guo et al., "Single path one-shot neural architecture search with uniform sampling," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2020, pp. 544–560.
[53] L. Yang and Q. Jin, "FracBits: Mixed precision quantization via fractional bit-widths," 2020, *arXiv:2007.02017*.
[54] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2014, pp. 740–755.
[55] XILINX. *7 Series DSP48e1 Slice User Guide*. Accessed: Jan. 28, 2023. [Online]. Available: https://docs.xilinx.com/v/u/en-U.S./ug479_7Series_DSP48E1
[56] XILINX. *Zynq Ultrascale+ MPSoC Power Advantage Tool 2018.1*. Accessed: Jan. 28, 2023. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841803/Zynq+UltraScale+MPSoC+Power+Advantage+Tool+2018.1
[57] XILINX. *ZCU102 Evaluation Board User Guide*. Accessed: Jan. 28, 2023. [Online]. Available: https://www.xilinx.com/support/documents/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf

Wonjae Lee received the B.S. degree from the Department of Electrical and Computer Engineering, Seoul National University, Seoul, South Korea, in 2019, and the M.S. degree from the Graduate School of Convergence Science and Technology, Seoul National University, in 2022.

Since 2022, he has been an Engineer with Samsung Electronics Company Ltd., Hwaseong, South Korea. His research interests include quantized neural network algorithms and systems, and low-power and resource-efficient hardware design for deep learning inference/training.

Kukbyung Kim received the B.S. degree in electrical engineering from Korea University, Seoul, South Korea, in 2009, and the M.S. degree from the Graduate School of Convergence Science and Technology, Seoul National University, Seoul, in 2022.

Since 2012, he has been a Senior Engineer with Hanwha Systems Company Ltd., Seongnam, South Korea, where he is currently working on embedded hardware development. His current research interests include object detection algorithms, hardware accelerators, field-programmable gate arrays (FPGAs), and low-power embedded systems.

Woohyun Ahn received the B.S. and M.S. degrees in electrical engineering from Chungbuk National University, Cheongju-si, South Korea, in 2012 and 2014, respectively.

Since 2015, he has been a Senior Engineer with Hanwha Systems Company Ltd., Seongnam, South Korea, where he is currently working on embedded software development. His current research interests include digital signal processing and embedded software implementation such as DSP, field-programmable gate arrays (FPGAs), and system-on-chip (SoC).

Jinho Kim received the B.S. and M.S. degrees in information and telecommunication engineering from Korea Aerospace University, Goyang-si, South Korea, in 2004 and 2006, respectively.

Since 2010, he has been the Chief Engineer of Hanwha Systems Company Ltd., Seongnam, South Korea, where he is currently working as a System Engineer. His current research interests include object detection algorithms, hardware accelerators, and system integration.

Dongsuk Jeon (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2009, and the Ph.D. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2014.

From 2014 to 2015, he was a Postdoctoral Associate with the Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently an Associate Professor with the Graduate School of Convergence Science and Technology, Seoul National University. His current research interests include hardware-oriented machine learning algorithms, hardware accelerators, and low-power circuits.