UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision

Jinmook Lee, *Student Member, IEEE*, Changhyeon Kim, *Student Member, IEEE*, Sanghoon Kang, *Student Member, IEEE*, Dongjoo Shin, *Member, IEEE*, Sangyeob Kim, and Hoi-Jun Yoo, *Fellow, IEEE*

***Abstract*—An energy-efficient deep neural network (DNN) accelerator, unified neural processing unit (UNPU), is proposed for mobile deep learning applications. The UNPU can support both convolutional layers (CLs) and recurrent or fully connected layers (FCLs) to support versatile workload combinations to accelerate various mobile deep learning applications. In addition, the UNPU is the first DNN accelerator ASIC that can support fully variable weight bit precision from 1 to 16 bit. It enables the UNPU to operate on the accuracy-energy optimal point. Moreover, the lookup table (LUT)-based bit-serial processing element (LBPE) in the UNPU achieves the energy consumption reduction compared to the conventional fixed-point multiply-and-accumulate (MAC) array by 23.1%, 27.2%, 41%, and 53.6% for the 16-, 8-, 4-, and 1-bit weight precision, respectively. Besides the energy efficiency improvement, the unified DNN core architecture of the UNPU improves the peak performance for CL by 1.15× compared to the previous work. It makes the UNPU operate on the lower voltage and frequency for the given DNN to increase energy efficiency. The UNPU is implemented in 65-nm CMOS technology and occupies the 4 x 4 mm^2 die area. The UNPU can operates from 0.63- to 1.1-V supply voltage with maximum frequency of 200 MHz. The UNPU has peak performance of 345.6 GOPS for 16-bit weight precision and 7372 GOPS for 1-bit weight precision. The wide operating range of UNPU makes the UNPU achieve the power efficiency of 3.08 TOPS/W for 16-bit weight precision and 50.6 TOPS/W for 1-bit weight precision. The functionality of the UNPU is successfully demonstrated on the verification system using ImageNet deep CNN (VGG-16).**

***Index Terms*—Bit serial, deep learning, deep learning ASIC, deep learning hardware, deep neural network (DNN), mobile deep learning.**

I. Introduction

RECENTLY, deep learning has been successful for many applications such as image classification or machine translation due to its high accuracy. Along with the advance of deep learning algorithm, there have been many efforts to apply deep learning to mobile applications such as face recognition, speech recognition, and user identification [1], [2]. However, deep neural networks (DNNs), components of deep learning algorithms, have over millions neurons and parameters, so general purpose graphics processing units (GPGPU) have been used for DNN acceleration which has huge computational complexity (over 30 GOPS per inference for VGG-16). However, GPGPUs cannot be used for power hungry mobile environment because of its high-power consumption over 100 W, thus, many algorithms have been developed to operate DNNs using mobile processors or embedded GPU [3]–[7].

They have tried to reduce computational complexity of DNNs by network quantization [8] or pruning [9] or approximations [10], but they still have limitation to implement real-time deep learning on mobile platform. For example, even though Kim *et al.* [11] used the compressed DNN, it could not satisfy real-time requirement and it showed still high-power consumption over 1 W.

To resolve this problem, many silicon-implemented DNN hardware accelerators optimized for DNN operation have been studied (Fig. 1). They were designed with the architecture that dedicated for DNN, so versatility of the chip is relatively low; however, their architecture optimized for DNN makes much higher power efficiency of DNN acceleration than that of GPGPU [15].

References [12] and [13] are ASICs for fully connected layers (FCLs) or recurrent layers (RLs) acceleration and have architecture optimized for energy-efficient matrix multiplication. References [14]–[19] were designed for acceleration of convolutional layers (CLs) with on-chip data reuse and large-scale integration of processing elements (PEs). In particular, [20] and [21] are representative ASICs that support both RLs/FCLs as well as CLs. However, [20] and [21] have the following limitations. In this paper, unified neural processing unit (UNPU) solves this problem with two main features.

A. Limitations of Previous Work

1) Limited Precision Scalability: Reference [20] supports 4, 8, and 16 bit and [21] supports 8 and 16 bit precision for DNN inference operation. However, optimal weight precision

Manuscript received May 6, 2018; revised July 6, 2018; accepted August 1, 2018. This paper was approved by Guest Editor Masato Motomura. This work was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT & Future Planning under Grant NRF-2015R1A2A1A05001889. *(Corresponding author: Jinmook Lee.)*

The authors are with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: jinmooklee@kaist.ac.kr).

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

Digital Object Identifier 10.1109/JSSC.2018.2865489

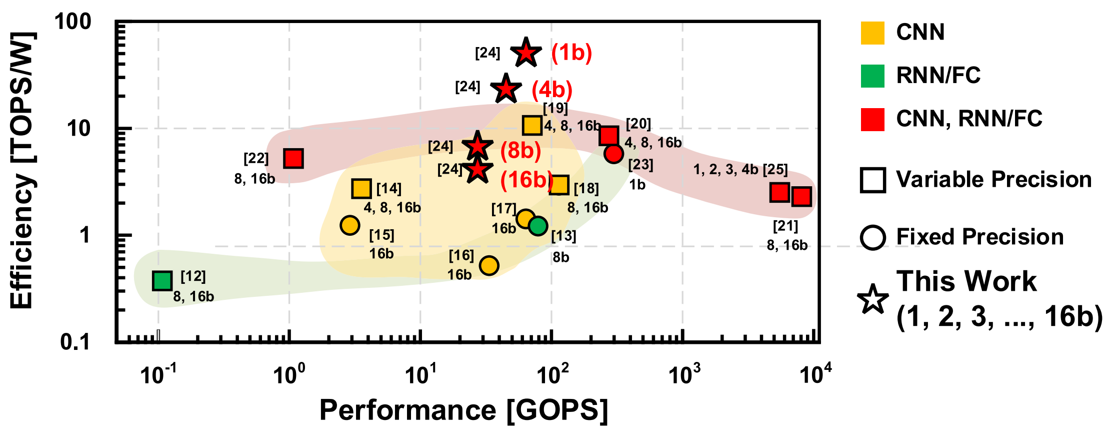


Fig. 1. Power efficiency of reported DNN accelerator ASICs. If any ASIC supports multi-bit precision, the best efficiency point is used for the plot.

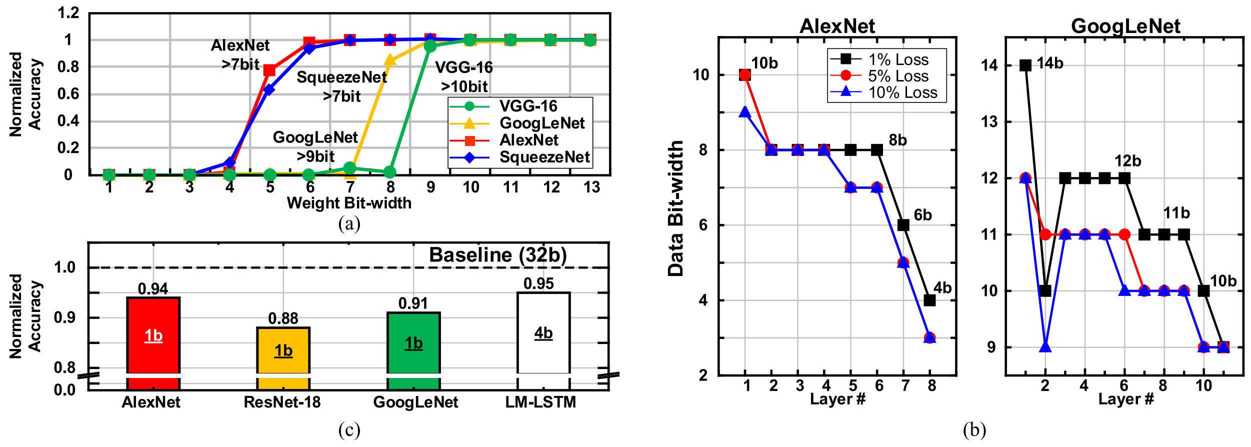


Fig. 2. (a) Different DNNs requires different optimal weight precision [26]. (b) Different layers in a DNN has different optimal data width [27]. (c) Different accuracy requirement of DNNs requires different weight precision [28]–[30].

may not be either 4 or 8 or 16 bit because of the following three reasons. First, as shown in Fig. 2(a), different DNNs have different optimal weight precision [26]. In other words, the minimum weight bit precision, which makes minimal accuracy loss of a given DNN, varies among different DNNs. Second, as described in Fig. 2(b), different layers in a given DNN have different optimal precision [27]. In other words, the data width that makes minimum accuracy loss is different among layers. Third, as shown in Fig. 2(c), the weight precision required for inference operation of DNNs depends on the desired accuracy of DNNs [28]–[30]. For example, the AlexNet shows only 6%p of accuracy loss even for 1-bit weight precision when it compared to 32-bit weight precision; however, the ResNet-18 shows accuracy loss of 12%p for 1-bit weight precision. The lower weight bit precision of DNNs makes the higher energy efficiency of DNN acceleration, and the desired weight precision of DNNs depends on the target accuracy of the given DNN. Therefore, fully variable weight precision has to be supported to accelerate DNNs on the energy-accuracy optimal point. However, [20] and [21] could not accelerate a given DNN at the energy-accuracy optimal point.

2) Separate CNN and RNN/FC Core Architecture: Reference [20] was designed with separate DNN core architecture for acceleration of both computation-dominant CLs and memory-dominant RLs (or FCLs) with a single accelerator chip. Specifically, convolution cores of [20] perform acceleration of CLs with massive parallelism and on-chip data reuse, and RNN cores perform acceleration of RLs (or FCLs) with weight quantization for reduction of external memory accesses. Although the PEs for RLs can be reconfigurable into PEs for CLs or vice versa, only partial reconfiguration was possible, resulting in marginal performance improvement.

In this paper, above two limitations of previous works are resolved by two features: 1) unified DNN core architecture and 2) fully variable weight bit precision. We present an UNPU supporting CLs, RLs, and FCLs with fully variable weight bit precision from 1 to 16 bit. As shown in Fig. 3, the reuse of input feature map is more efficient than the reuse of weights under low-weight bit precision and the operations of CLs become identical to those of RLs and FCLs when the input feature maps of the CLs are vectorized into a 1-D vector. It makes the hardware can be fully shared in the UNPU for CLs and RLs (or FCLs) and makes 1.15× higher peak performance of the UNPU than [20]. Moreover, the lookup table-based bit-serial PE (LBPE) is designed in the UNPU for energy-accuracy optimal DNN operations with variable weight bit precision from 1 to 16 bit through iterations of 1-bit weight multiply-and-accumulate (MAC) operation as explained in Fig. 4. Furthermore, an aligned feature map

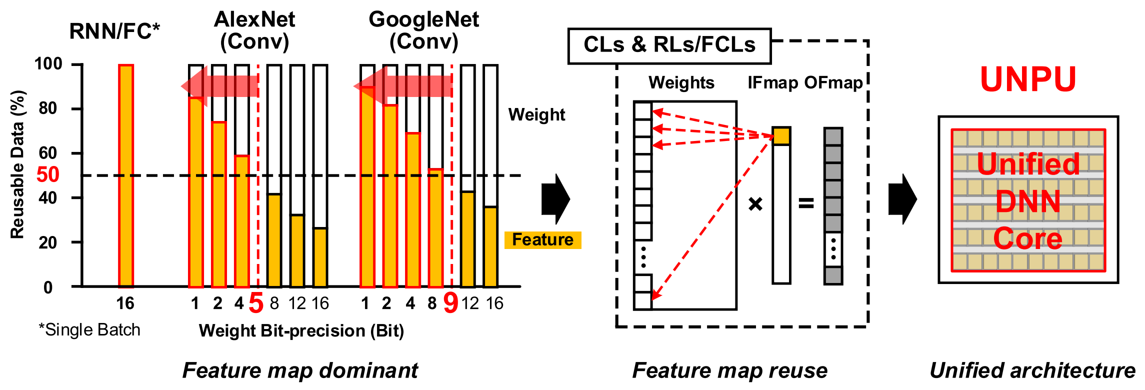


Fig. 3. Unified data path for CLs and RLs/FCLs in the UNPU by reusing feature map during DNN operation.

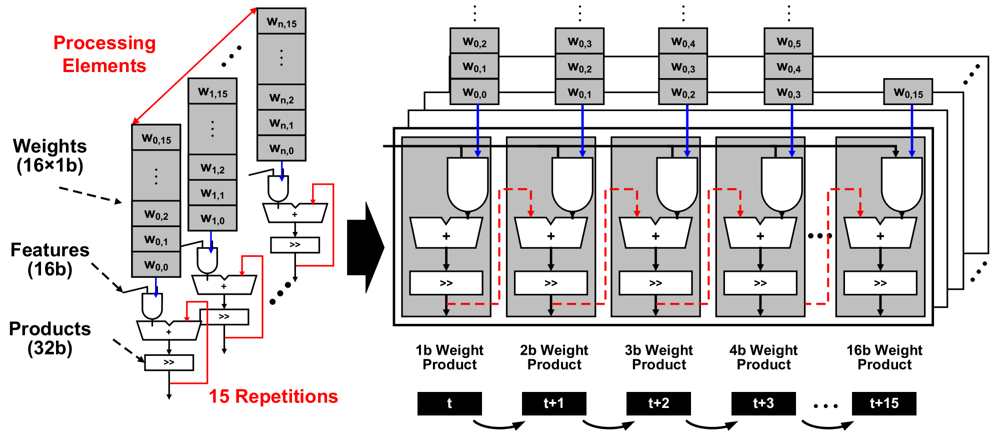


Fig. 4. UNPU supports fully variable weight from 1 to 16 bit precision with bit-serial MAC operation.

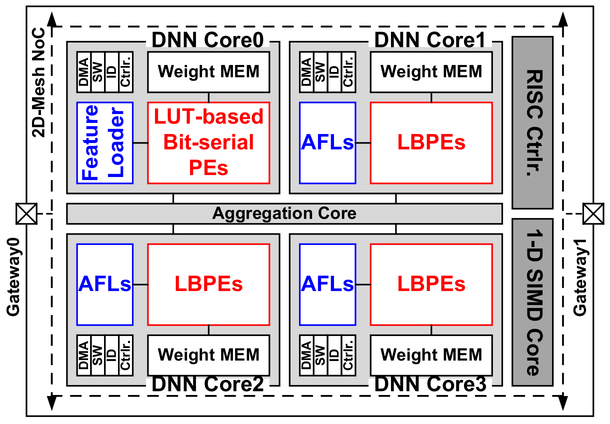
loader (AFL) is designed to minimize the amount of external memory accesses required for input feature map fetching by exploiting the data locality among convolution operations and re-configure the UNPU between CLs and RLs.

3) Comparison With Related Works: A related work, bit fusion [31] presented after the UNPU [24], is designed with bit-serial processing. However, the key feature of bit fusion, spatio-temporal fusion processing, is already supported in UNPU. UNPU performs temporal processing with bit-serial fashion and spatial-processing using spatially distributed parallel PEs whose operands are multiple input activations. In addition, when it comes to the supported bit precision, UNPU can support fully variable precision from 1 to 16 bit, unlike bit fusion whose supported precision is either only 2 or 4 or 8 or 16 bit. The optimal bit precision of DNNs can be any bit between power of 2, therefore, bit-scalability of bit fusion is insufficient to accelerate DNNs on the energy-accuracy optimal point. Another related work, stripes [27] was designed with bit-serial processing element as well. However, PEs of stripes was implemented in a way of straightforward bit-serial arithmetic, and it causes 32% area overhead compared with fixed-point arithmetic design. In this paper, UNPU optimized the area overhead and improved power efficiency using lookup table-based PEs (LBPEs). The LBPEs in UNPU shows the energy consumption reduction by 23.1%, 27.2%, 41%, and 53.6% for the case of 16, 8, 4, and 1 bit, respectively, compared with fixed-point arithmetic units.

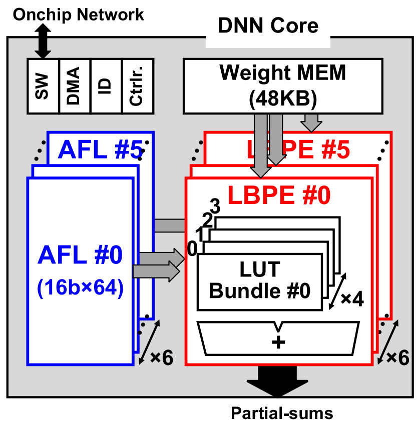
The rest of this paper is organized as follows. In Section II, the architecture of the UNPU and unified DNN core in the UNPU is described. Section III explains how to map workload to the unified DNN core for CLs and RLs (or FCLs) operation. Section IV describes the architecture and operation of the LBPE. Section V explains the architecture and operation of the AFL in the unified DNN core of the UNPU. Section VI shows the chip implementation results. Section VII includes discussion and Section VIII provides the conclusion.

II. Overall Architecture

Fig. 5(a) explains the overall architecture of the UNPU. The UNPU includes 2-D mesh type network-on-chip (NoC), four unified DNN cores, 1-D SIMD core, RISC controller, aggregation core, and two external gateways connected to this NoC. The four unified DNN cores independently generate partial sums, which are transferred to the aggregation core through a dedicate path and used to calculate final output feature maps. The 1-D SIMD core performs vector processing such as non-linear activation function or element-wise multiplication to make final output activation. The activation function is implemented using piecewise linear approximation. The RISC controller is used to execute instructions needed to perform inter-core data communication via the NoC during the DNN operation. The external gateways connect the UNPU to the external memory, and they are used to load the weight and activation into the UNPU, or write back the intermediate



(a)



(b)

Fig. 5. (a) Overall architecture of the UNPU, it has four DNN cores and they are connected to 2-D mesh NoC. They make partial-sums and transfer partial-sums to aggregation core. (b) Each DNN core includes six pairs of AFL-LBPE.

partial sums generated during the DNN operation to the off-chip storage.

A. Detail Architecture of the Unified DNN Core

Fig. 5(b) illustrates the architecture of the unified DNN core. The unified DNN core includes a switch and a DMA controller connected to the NoC for inter-core data communication, custom instruction decoder, and controller for custom instruction set. In addition, there are six AFL-LBPE pairs with a 48-KB local weight memory. The DNN weights are broadcasted from the local weight memory to six LBPEs.

The LBPE is a PE that performs matrix multiplication for DNN operation. The LBPE performs MAC operation using lookup table (LUT) for energy-efficient matrix multiplication and supports MAC operation with every weight bit precision from 1 to 16 bit in a bit-serial manner. For example, the UNPU can perform 18,432 1-bit weight MAC operations per cycle and 13,824 16-bit weight MAC operations per 16 cycles in parallel using the LBPEs. Four LUT bundles are included in the LBPE, and each LUT bundle is used for MAC operations by accessing LUT in it. The LBPE generates partial sums of DNN output feature map by accumulating the partial sums obtained from the four LUT bundles.

The AFL works in pairs with the LBPE. The AFL is a buffer that transfers the input feature maps required for the LBPE after aligning input feature maps. The alignment enables the LBPE to operate DNNs regardless of whether the network currently accelerated is CL or an *RL* (or FCL). The UNPU accelerates the CLs and RLs (or FCLs) with the unified DNN core by reusing input feature maps multiple times.

III. Workload Mapping on Unified DNN Core

Figs. 6 and 7 explain how workloads are mapped to the unified DNN core in the UNPU to accelerate CLs and RLs (or FCLs), respectively. In order to allocate workloads of different operation patterns to the UNPU, the UNPU computes partial sums of DNN based on feature map reuse. In addition, the AFL, which plays a role of data feeding to the unified DNN core, gives abstraction to LBPEs in the DNN core to enable the same operation regardless of the kind of network to be accelerated. More detail architecture and operation of the AFL will be covered in the Section V, and this section will focus on how the DNN core works.

A. Case of Recurrent Layers (or Fully-connected Layers)

Computation of basic FCLs is performed as the following equation:

$$y = f(Wx + b) \tag{1}$$

where x is input feature vector, W is weight matrix, b is bias, and f is activation function for a given model.

Computation of basic recurrent neural network is performed as the following equation:

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1}) \tag{2}$$
$$y_t = g(W_{hy}h_t) \tag{3}$$

where, W_{hx}, W_{hh}, and W_{hy} are weight matrices and x and h are input feature vector and hidden state vector of the given RNN model, respectively.

From (1)–(3), we can notice that the FCLs and RLs are composed of matrix-vector product (e.g., $W_{hx}x_t$) and element-wise operations such as the activation function. The gated recurrent unit RNN and the long short-term memory RNN are also composed of these two key operations. In this situation, if an arbitrary weight matrix has a dimension of M by N and the input feature vector to be multiplied has a dimension of N by 1, the ratio of matrix-vector product to the element-wise operation of input feature vector becomes approximately M:1. Therefore, it is very important to perform the matrix-vector product efficiently for FCLs or RLs acceleration. The UNPU uses the LUT to perform an energy-efficient matrix-vector product, and remaining operations such as element-wise multiplication or non-linear activation functions are performed using the 1-D SIMD core and the RISC controller.

Since the weight matrix of RLs (or FCLs) has size of over 1 MB (e.g., weight matrix of 1024 by 1024 dimension with 16 bit precision is 2 MB) and each weight value is used only once, it is not suitable to store whole weight matrix into the on-chip memory. Therefore, it is efficient to load a part of weight matrix into the chip and then reuse the input feature

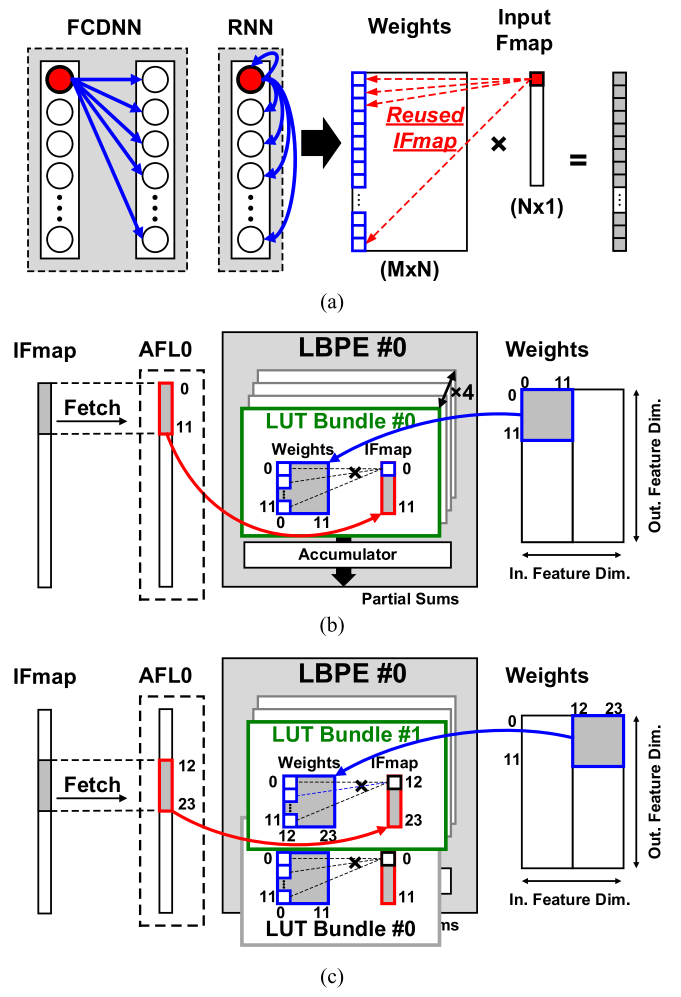


Fig. 6. Workload mapping on the unified DNN core for RLs/FCLs operation. (a) Feature map reuse for RLs/FCLs. (b) 1-D IFmap moves to LUT bundle and LUT bundle reuse this to make partial-sums. (c) Each LUT bundle is interleaved, thus successive IFmaps are assigned to the successive LUT bundle.

map to be multiplied with these values. In the UNPU, as shown in Fig. 6(a), the related weights for the given input feature map are loaded at first, then partial sums are calculated until the complete product is obtained.

For example, an 1-D input feature vector of RLs (or FCLs) is split into 12 elements and loaded into the LUT bundle in the LBPE as described in Fig. 6(b). The LUT bundle is a PE that performs MAC operations using given input feature vector. After the input feature vector is loaded, weight values of 12 by 12 dimensions are loaded from the local weight memory to the LUT bundle. Then, the LUT bundle calculates partial sums of matrix (12 by 12)-vector (12 by 1) product. At the same time, since the elements of input feature vector are interleaved, the following 12 input feature elements are assigned to the next LUT bundle and used to calculate partial product as explained in Fig. 6(c). One LBPE includes four LUT bundles, and partial sums calculated from them are accumulated in the LBPE to calculate complete output feature map of RLs (or FCLs).

Multiple input feature maps that is obtained from different batches are assigned to the different LBPEs in the

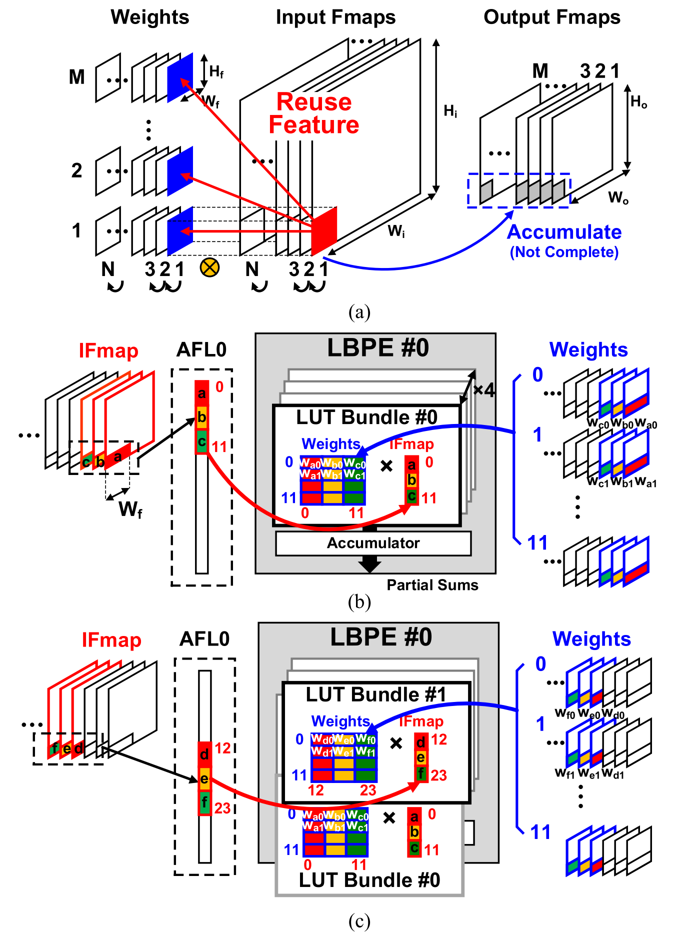


Fig. 7. Workload mapping on unified DNN core for CLs operation. (a) Feature map reuse for CLs operation. (b) 2-D IFmaps are converted to 1-D vector and moves to LUT bundle. (c) Each LUT bundle is interleaved as like the case of RLs/FCLs.

unified DNN core. Batch-level parallelism is possible, because LBPEs in the same DNN core share weight values, which are broadcasted from the local weight memory.

B. Case of Convolutional Layers

Fig. 7 explains how to map CLs to the UNPU. Basic operation sequence for CLs is the same as RLs (or FCLs), the feature map reuse. For example, as shown in Fig. 7(a), an image patch from a channel of the input feature maps is multiplied and accumulated with multiple patches from multiple weight maps (total M channels). Then, it makes partial sums corresponding multiple channels of the output feature maps. Because these partial sums are not complete values, we need to move to the next channel of the input feature maps (total N channels) and repeat MAC operations to obtain the complete output feature map.

The main difference of CLs compared to RLs (or FCLs) is that the input feature maps and weights have 2-D shape. The CLs can be accelerated with exactly same architecture that is used for RLs (or FCLs), if only the PEs in the UNPU does not need to care whether the current operation is for CLs or RLs (or FCLs). In order to give abstraction to the PEs in the UNPU, 2-D input feature maps of CLs should be converted to 1-D

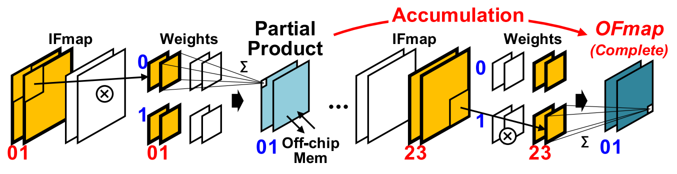


Fig. 8. Weight and partial-sum management for CLs in the UNPU.

shape to make the same shape as RLs (or FCLs), and the local weight memory should store the required weight values with the order required for CLs operation (bit serially).

Fig. 7(b) explains an example of CLs operation. For CLs with filter width of W_f, the UNPU makes a 1-D input feature vector by the AFL, which consists of input activations from multiple channels. The reason why the input activations from multiple input channels are concatenated into a 1-D vector is each input activation of RLs (or FCLs) corresponds to each channel of input feature maps of CLs. When the input feature maps are concatenated in this order for the CLs operation, the order in which the output feature maps computed becomes the same as RLs (or FCLs). Therefore, not only architecture for the matrix-vector product but also architecture for the partial-sum accumulation can be shared between CLs and RLs (or FCLs).

After constructing the 1-D input feature vector, it is divided into 12 elements and moves to the LUT bundles in the LBPE. The corresponding weight values stored in the local weight memory moves to the LUT bundles. Then, the LUT bundle calculates partial sum of product between a matrix of 12 by 12 dimensions and a vector of 12 by 1 dimensions. The calculated partial sum corresponds to 12 different channels of output feature map. As like the RLs (or FCLs), the 1-D input feature vector is interleaved to the LBPEs as described in Fig. 7(c). The partial sums computed from the LUT bundles are accumulated in the LBPE, and then used to calculate the final output feature map.

The only difference from RLs (or FCLs) in CLs operation is that weights are reused within the same input feature map channel due to the nature of convolution operation (convolutional weight reuse). Therefore, the slided input feature maps in the same channel are assigned to six LBPEs in the DNN core to exploit convolutional weight reuse for six LBPEs.

By the help of the unified DNN cores, the peak performance for CLs operation of the UNPU is increased by 1.15× than [20] that has the heterogeneous core architecture and same die area. It is because that the UNPU has the increased number of PEs for CLs operation, and it enables the UNPU to operate the given DNNs at the lower supply voltage and operating frequency to increase power efficiency.

C. Weight and Partial-Sum Management for CLs

Fig. 8 explains how to manage weights and partial sums to calculate CLs with limited on-chip memory in the UNPU. In the feature-map-reuse-based CLs computation, if the all of weights of target layer cannot be stored in the on-chip memory, the weights must be redundantly re-loaded into the chip. Instead, the UNPU divides the weight maps into multiple segments that can be stored entirely on-chip memory, and then loads only one segment into the UNPU. This is similar to the channel division method of [20]. By using the loaded segment of weight maps, the UNPU calculates partial sums of output feature map; however, they are not complete values. Therefore, after the output feature map moves out to the off-chip storage, load the next weight segment into the UNPU and calculate partial sum and accumulate it with previously calculated partial sums to obtain complete output feature map. Although this approach increases the amount of EMA for feature map, it reduces the redundant EMA for weight load. In the UNPU, the number of weight segments is decided layer-by-layer to minimize total EMA considering the size of weight maps and feature maps.

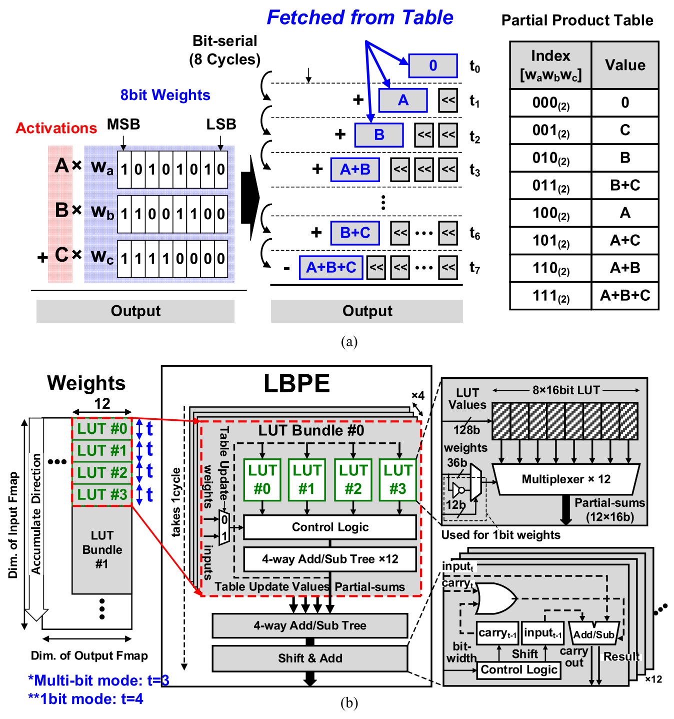


Fig. 9. (a) Concept of LBPE. (b) Detail architecture of LBPE.

IV. Lookup Table-Based Bit-Serial Processing Elements

A. Concept of LBPE

Fig. 9 explains the detail architecture and operation of the LBPE. Key concept of LBPE is related to feature-map-reuse-based DNN computation and bit-serial MAC operation. For example, as shown in Fig. 9(a), when three activations A, B, and C are multiplied with corresponding weights, these activations are reused many times with different weights. Since the LBPE operates bit serially, MAC operations with weights of N-bit precision are calculated sequentially from the LSBs of the weights to the MSBs for N cycles. In this situation, since the input activations are fixed during bit-serial MAC operation, the number of possible partial-product values is predetermined. The LBPE makes a table including these partial products to perform bit-serial MAC operations. In the above example, the number of table entries in an LUT becomes eight because the number of possible all of partial products for given three input activations (A, B, and C) is eight (0, A, B, C, A + B, A + C, B + C, and A + B + C).

B. Detail Architecture of LBPE

Fig. 9(b) explains detail architecture of the LBPE. The LBPE includes four LUT bundles, four-way add/sub tree, and shift-and-add logic. Each LUT bundle includes four LUT modules, control logic, and four-way add/sub trees. The control logic in the LUT bundle makes four-way add/sub trees calculate table update values when new input activations arrive at the LBPE. The partial sums computed from the four LUT bundles are accumulated using four-way add/sub tree included in the LBPE, and the shift-and-add logic performs bit-serial MAC operation by shifting and accumulating the partial sums.

Each LUT module makes twelve 1-bit weight partial products by using 12 multiplexers. As an example, for multi-bit weight MAC operation, each LUT module is updated with all possible partial products of three input activations. Then, 36 bits from weight matrix of 3×12 dimension (the number of activations $\times$ the number of multiplexers) are used as indices of LUT module for every cycle to calculate 12 partial products simultaneously. For 1-bit weight MAC operation, LUT is updated using four input activations. Then, 48 bits from weight matrix of 4×12 dimension (the number of activations $\times$ the number of multiplexers) are used as table indices. The reason why the number of input activations differs between 1-bit mode and multi-bit mode will be explained in Section IV-C. The partial products calculated from the four LUT modules are summed, and the resulting values are summed again using four-way adder tree in LBPE.

To sum up, one LUT bundle (LUT module $\times$ 4) can cover 12×12 region of weight matrix and 1 LBPE (LUT bundle $\times$ 4) can cover 48 $\times$ 12 region of weight matrix. For example, for the MAC operation of 16-bit weight precision, one LBPE can calculate 48 $\times$ 12 MAC operations for 16 cycles.

C. Two Modes of LBPE

The LBPE can support matrix-vector products with variable weight bit precision from 1 to 16 bit using bit-serial operation. When the LBPE performs MAC operations, operation sequence depends on whether the weight bit precision is 1 or multi-bit (2–16 bit). An LUT module can perform three-way bit-serial MAC for the multi-bit weight (2–16 bit); however, it can perform four-way bit-serial MAC for 1 bit weight by using the 2's complement principle. Fig. 10 explains how the input activations and corresponding weight matrix are mapped to the LUT module for the above two cases.

1) Multi-Bit Weight Precision Mode (2–16 bit): For this case, an LUT module includes all of possible partial products for three input activations (A, B, and C) such as 0, A, B, C, A + B, A + C, B + C, and A + B + C. In this situation, 12 multiplexers connected to the LUT uses each bit of weights as table indices. For example, at the first cycle of bit-serial MAC operation, 12 multiplexers make partial product by using the 0th bit (LSB) from $\{w_{0,0}, w_{1,0}, w_{2,0}\}$, $\{w_{0,1}, w_{1,1}, w_{2,1}\}, \ldots, \{w_{0,11}, w_{1,11}, w_{2,11}\}$. It makes 12 partial sums corresponding to the following matrix-vector product,

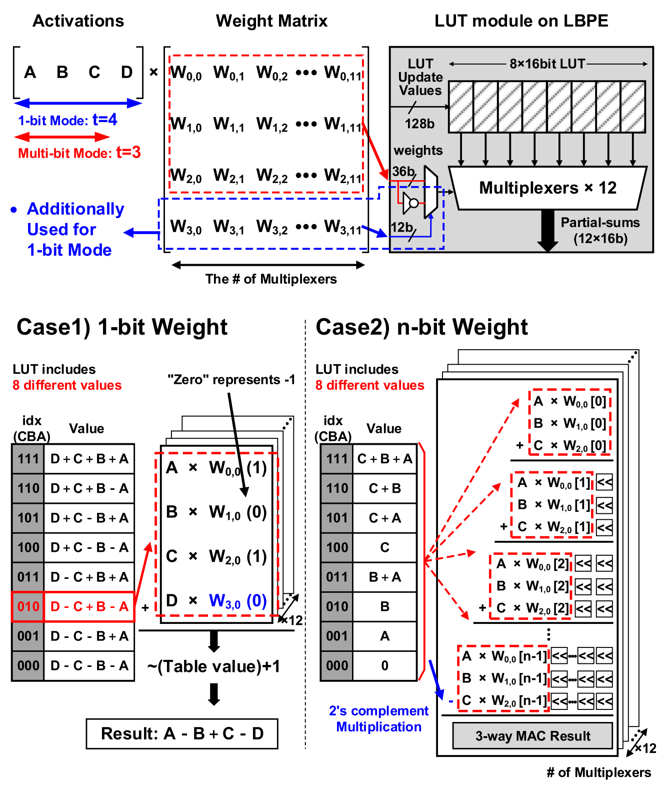


Fig. 10. Two modes of LBPE for 1- and multi-bit weight precision.

where $w_{a,b}$ [n] is nth bit of $w_{a,b}$ and each of $w_{a,b}$ [n] has value of 0 or 1

$$\text{Cycle1}: \begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} w_{0,0}[0] & w_{0,1}[0] & \cdots & w_{0,11}[0] \\ w_{1,0}[0] & w_{1,1}[0] & \cdots & w_{1,11}[0] \\ w_{2,0}[0] & w_{2,1}[0] & \cdots & w_{2,11}[0] \end{bmatrix}. \tag{4}$$

At the second cycle, second bits of weights $\{w_{0,0}[1], w_{1,0}[1], w_{2,0}[1]\}$, $\{w_{0,1}[1], w_{1,1}[1], w_{2,1}[1]\}, \ldots, \{w_{0,11}[1], w_{1,11}[1], w_{2,11}[1]\}$ are used as table indices, and 12 partial sums are fetched from LUT. Then, 12 partial sums from the second cycle are accumulated with 12 partial sums from the first cycle after a left shift operation. After N iterations of this process for N cycles, twelve three-way MAC operations with N-bit weight precision are done.

To sum up, an LUT module in multi-bit mode can hold all of possible partial products of three input activations, thus an LUT module can make 3 $\times$ 12 bit-serial operations per cycle (twelve three-way bit-serial MAC operations). Therefore, an LUT bundle, accumulating partial products from four LUT modules, performs 12×12 bit-serial operations per cycle, as described in Figs. 6 and 7.

2) 1-bit Weight Precision Mode: For binary weights (1-bit precision), the MAC operation is replaced with additions and subtractions since weight values are either 1 or -1. In this situation, an LUT module can perform four-way bit-serial MAC operation of four input activations (A, B, C, and D) with 2's complement principle. As shown in Fig. 10, eight partial products whose values are A + B + C + D, $-$A + B + C + D, ..., and $-$A $-$ B $-$ C + D are

TABLE I
SUMMARY OF PARALLELISM IN UNPU

	LUT Modules	LUT Bundles	LBPEs	DNN Cores	UNPU
RLs (FCLs)	Different input activations	Accumulate Psums from 4 LUT modules	Accumulate Psums from 4 LUT bundles	6 Different Batches	Accumulate Psums from 4 DNN Cores
CLs				6 Different Windows	
# of bit-serial Ops per cycle (Multi-bit)	3×12	3×12×$\underline{4}$	3×12×4×$\underline{4}$	3×12×4×4×$\underline{6}$	3×12×4×4×6×$\underline{4}$
# of bit-serial Ops per cycle (1-bit)	4×12	4×12×$\underline{4}$	4×12×4×$\underline{4}$	4×12×4×4×$\underline{6}$	4×12×4×4×6×$\underline{4}$

calculated and stored in the LUT. Then 48 bits from weights $\{w_{0,0},\ldots,w_{3,0}\},\{w_{0,1},\ldots,w_{3,1}\},\ldots,\{w_{0,11},\ldots,w_{3,11}\}$ are used as indices for table access, where $w_{a,b}$ has either 0 or 1, and 0 means a subtraction and 1 means an addition. For example, in Fig. 10, $w_{0,0}$, $w_{1,0}$, $w_{2,0}$, and $w_{3,0}$ are 1, 0, 1, and 0, respectively. Therefore, the desired partial sum is A − B + C − D. To obtain this, −A + B − C + D is fetched from the table using $w_{0,0}$, $w_{1,0}$, $w_{2,0}$ as indices, and then it is negated because $w_{3,0}$ is 0.

3) Summary of Parallelism: Table I explains how the parallelism hierarchy in the UNPU is for two modes of LBPE. First, each LUT module has parallelism in the level of different input activations. For multi-bit mode, eight table entries in an LUT module can hold all of possible partial products for three input activations, thus the number of bit-serial operations per cycle is 3 × 12. Twelve is number of multiplexers in an LUT module. For 1-bit mode, all of possible partial products for four input activations can be calculated just eight table entries in an LUT module. Therefore, the number of bit-serial operations for 1-bit mode becomes 4 × 12. LUT bundles accumulate partial sums from four LUT modules, and resulting values from LUT bundles are accumulated again in LBPEs. DNN cores perform batch-wise parallelism for RLs and parallelism on different windows for CLs. UNPU accumulates partial sums from DNN cores using aggregation core.

D. Comparison and Analysis of Energy Consumption With Conventional Fixed-Point MAC Array

Fig. 11 shows energy consumption comparison between the LBPE and conventional fixed-point MAC array. The comparison is performed using an LBPE and fixed-point MAC array whose throughput is the same with an LBPE. For example, the LBPE can calculate 12 × 48 MAC s per 16 cycles for the MAC operations with 16 bit weights. Therefore, a fixed-point MAC array of 36 MACs/cycle is used for the comparison. The energy consumed for the LUT update of LBPE operation is included in the comparison. Input operands corresponding input activations are reused 1024 times for both the LBPE and the fixed-point MAC array. The LBPE shows the energy consumption reduction by 23.1%, 27.2%, 41%, and 53.6% for the case of 16, 8, 4, and 1 bit, respectively.

The reason of why the energy consumption gain of the LBPE increases at the lower weight bit precision is the power consumption reduction of the fixed-point MAC array is less than half when the weight bit precision becomes half. In other words, the fixed-point MAC arrays in the comparison groups

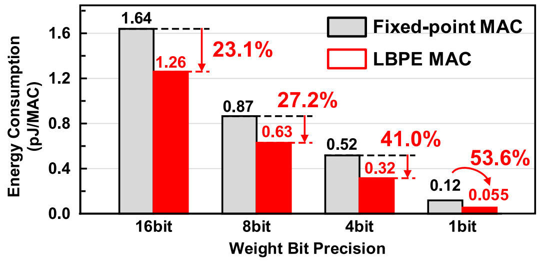


Fig. 11. Energy consumption comparison with fixed-point MAC array.

have the increased throughput when the weight bit precision is reduced such as 36 MACs/cycle (16 bit), 72 MACs/cycle (8 bit), and 144 MACs/cycle (4 bit). In this situation, although the bit precision of the weight is reduced, the bit precision required for partial-sum accumulation is not reduced, and the number of stages required for accumulation is increased due to the increased throughput. Therefore, the power consumption reduction of fixed point MAC is less than half. However, the power consumption of LBPE does not vary with different weight bit precision. Therefore, energy consumption gain of the LBPE increases when the weight bit precision is reduced.

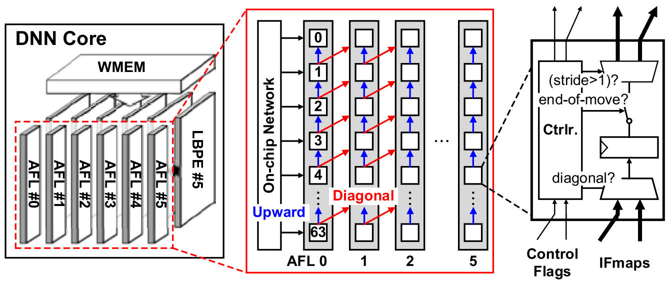
V. ALIGNED FEATURE MAP LOADER

A total of six AFLs are included in the unified DNN core, and each of them pairs with each LBPEs. AFLs have a role to transfer input feature map used for MAC operations to LUT bundles in the LBPE. In other words, the UNPU loads 1-D vectorized input feature vector into the AFLs and then transfer them to the LUT bundles to use them for DNN operation. An important feature of the AFL is that it has architecture that can align the input feature vector with desired order, which are necessary for the CLs operation.

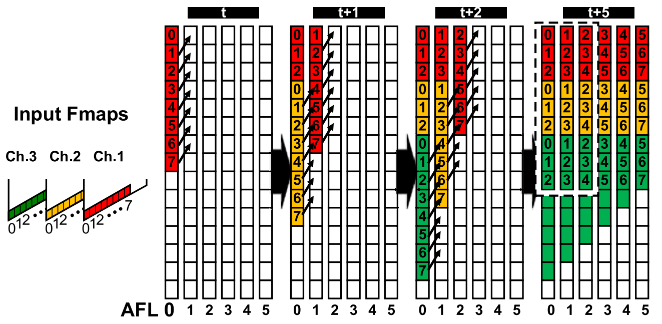
As shown in Fig. 12(a), the AFL has connections between AFLs and connections inside the AFL. These connections are used for alignment of input feature vector for the CLs operation. After the input feature vector is aligned by the AFL, the UNPU reuses these values several times to perform the DNN operation (feature map reuse). Therefore, the alignment operation of the AFL is performed only once when the new input feature vector arrives.

A. Detail Architecture of the AFL

As described in Fig. 12(a), an AFL consists of 64 sub-elements and each element includes control logic and multiplexers to control the movement of input activations,



(a)



(b)

Fig. 12. (a) Detail architecture of AFL. (b) Operation example of AFL (3 × 3 kernel and stride 1).

which are stored in the registers in itself. The two multiplexers in each element determine in which direction to receive the input activation as input and in which direction to transfer the input activation, respectively. The upward connection inside the AFL is architecture to make the UNPU support CLs with various strides, and diagonal connection between AFLs is architecture to make input activations align that are used for 2-D convolution operation. The control logic not only performs data movement in diagonal direction or upward direction, but also detects when the movement should end and makes the input activation no longer move.

B. Operation of Aligned Feature Map Loader

Fig. 12(b) explains how the AFL performs the alignment of input activations used for CLs operation with the unified DNN core, as described in Section III. It indicates the case that the UNPU operates CLs with kernel width 3 and stride 1.

At the first cycle, eight activations from Ch. 1 $(0, 1, \ldots, 7$ from Ch. 1) are loaded on the AFL 0. At the next cycle, the seven activations in the AFL 0 except the top-most one $(1, 2, \ldots, 7$ from Ch. 1) are shifted diagonally to the AFL 1, while the new eight activations from Ch. 2 $(0, 1, \ldots, 7$ from Ch. 2) are concatenated below the remaining three activations of Ch. 1 (0, 1, 2 from Ch. 1) in the AFL 0. At the third cycle, eight activations from Ch. 3 $(0, 1, \ldots, 7$ from Ch. 3) are concatenated below the remaining three activations of Ch. 2 (0, 1, and 2 from Ch. 2) on the AFL 0, while six activations in the AFL 1 from Ch. 1 $(2, 3, \ldots, 7$ from Ch. 1) are shifted diagonally to the AFL 2, and seven activations from Ch. 2 $(1, 2, 3, \ldots, 7$ from Ch. 2) are shifted diagonally to concatenate below the remaining three activations from Ch. 1 in the AFL 1. For the CLs with a larger kernel width, the UNPU divides the input feature vector with (kernel-width) + (stride) × 5 and load it into the AFL. In order to support various stride greater than 1, the AFL performs multiple upward shifts before transfer the activations to the next AFL with diagonally shift.

Architecture of AFL makes the UNPU load input activations efficiently. Input activations, reused during 2-D convolution (convolutional reuse), can move to the next AFL through diagonal connections in AFL to use them in the next AFL-LBPE pair. Then, the next AFL-LBPE pair will calculate the output feature map of the next pixel position. Summing up, as input data buffers for LBPEs, AFLs can align input activations with required data order for CLs and RLs. In addition, connections of AFL enable the UNPU to support various kernel size and stride along with convolutional data reuse of input activations.

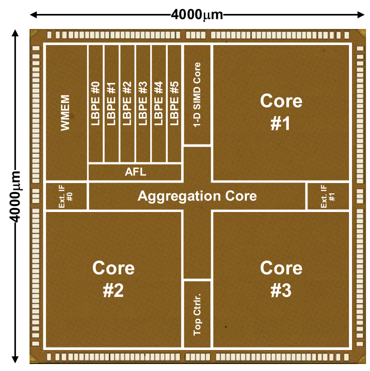
VI. Implementation Results

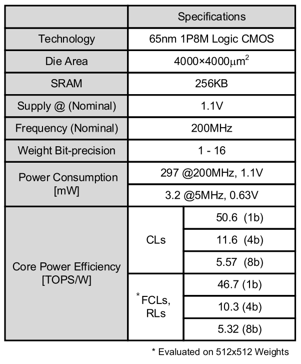
Fig. 13 shows the chip photograph and the performance summary of the UNPU fabricated with 65-nm 1P8M CMOS process. The UNPU occupies 4×4 mm^2 die area and supports fully variable weight bit precision from 1 to 16 bit with bit-serial operations. It consumes 297 mW at 200 MHz, 1.1 V and 3.2 mW at 5 MHz 0.63 V. Each component in the UNPU is designed with independent frequency domain. Therefore, the UNPU is possible to vary the operating frequencies of the unified DNN core, the aggregation core, and the NoC link in the UNPU independently for variable weight bit precision. The UNPU shows core power efficiency of 50.6 TOPS/W for CLs (5 × 5 kernel, 512 channel) with 1-bit weight bit precision, and 46.7 TOPS/W for FCLs with (512 × 512 weight matrix) 1-bit weight bit precision.

Fig. 13 shows the voltage-frequency and bit-width scaling result of the UNPU. The UNPU operates from 0.63- to 1.1-V supply voltage with a maximum 200-MHz operating frequency. The core power efficiency, as measured on CLs (5 × 5 kernels) with consideration of PE utilization is 3.08, 11.6, and 50.6 TOPS/W for the case of 16-, 4-, and 1-bit weight precision, respectively. When it is compared to multi-bit weight precision, the UNPU shows much better power efficiency at 1-bit weight precision. It is because the UNPU can operate 1.33× more MAC operations with 1-bit weight precision using 2's complement principle. Fig. 14 describes the power breakdown result of the UNPU.

Tables II and III show the VGG-16 benchmark results and performance comparison of the UNPU. Since the UNPU accelerates CLs in the way of reusing feature map, it has higher active PE percentage with layers of large channel depth, such as layer 12, or layer 13. In order to increase the UNPU efficiency in the layer 1, 2, or 3, where the channel depth is relatively small, the UNPU uses image division methodology presented at the [20]. If the UNPU cannot hold whole weight parameters required for calculation of the given layer, the weight and partial sum management methodology described in Section III is used to reduce external memory access required for weight fetching. When the UNPU is compared with the previous CNN accelerators, it shows 2.35× higher power efficiency than that of [19].

Tables IV and V show the AlexNet benchmark and performance comparison of the UNPU. The basic operation is





	Specifications	
Technology	65nm 1P8M Logic CMOS	
Die Area	4000×4000μm^2	
SRAM	256KB	
Supply @ (Nominal)	1.1V	
Frequency (Nominal)	200MHz	
Weight Bit-precision	1 - 16	
Power Consumption [mW]	297 @200MHz, 1.1V	
	3.2 @5MHz, 0.63V	
Core Power Efficiency [TOPS/W]	CLs	50.6 (1b)
		11.6 (4b)
		5.57 (8b)
	*FCLs, RLs	46.7 (1b)
		10.3 (4b)
		5.32 (8b)

* Evaluated on 512x512 Weights

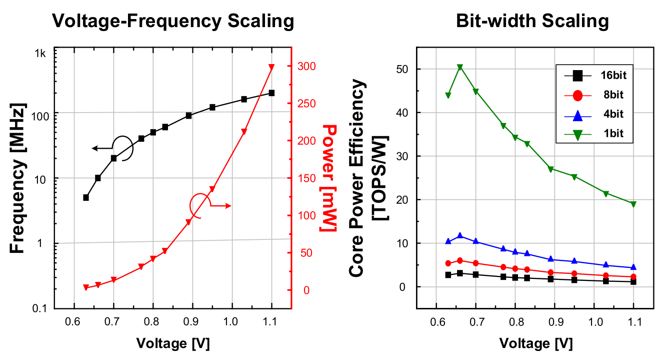


Fig. 13. Chip photograph and performance summary.

TABLE II
BENCHMARK ON VGG-16 (CONV)

8bit(F)x8bit(W)	Conv. Layers												
	1	2	3	4	5	6	7	8	9	10	11	12	13
The number of Operations (10^9)	0.17	3.70	1.85	3.70	1.85	3.70	3.70	1.85	3.70	3.70	1.21	0.92	0.92
Active PE (%)	77	93	93	95	95	100	100	100	100	100	100	100	100
External Memory Access (MB)	6.31	10.3	4.16	5.25	3.86	6.20	6.20	5.98	12.0	12.0	5.40	4.68	4.68

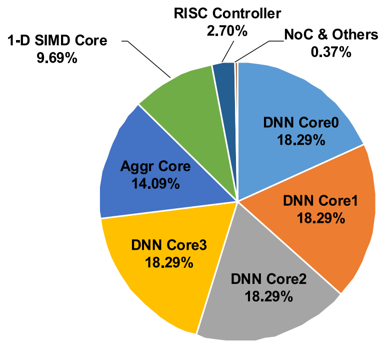


Fig. 14. Power breakdown of UNPU.

TABLE III
VGG-16 BENCHMARK COMPARISON

	[16]	[19]	This Work	
Frame Rate (fps)	0.7	1.67	0.97	18.3
Power Consumption (mW)	236	26	6.4	297
Power Efficiency (TOPS/W)	0.092 (Avg.)	2.0 (Avg.)	4.71(1)	1.91(2)
Bit Precision (bit)	16	N/A	8	8

(1) Core 10MHz, Link 15MHz, 0.7V
(2) Core 200MHz, Link 200MHz, 1.1V

similar to the case of VGG-16; however, the AlexNet has little accuracy loss even for the 1-bit weight precision ($<1\%$). In this case, the UNPU shows 27.3 TOPS/W power efficiency with 158 fps throughput.

TABLE IV
BENCHMARK ON ALEXNET

	Weight Precision	Conv. Layers				
		1	2	3	4	5
The number of Operations (10^6)	Common	211	448	299	224	149
Active PE (%)		77	93	100	96	96
External Memory Access (MB)	8bit	0.77	0.89	1.59	0.97	0.64
	1bit	0.74	0.45	0.35	0.23	0.18

TABLE V
ALEXNET BENCHMARK COMPARISON

	[19]	[20]	[22]	This Work		
Frame Rate (fps)	47	177	105	19.7	346	158
Power Consumption (mW)	44	63	290	6.1	290	7.7
Power Efficiency (TOPS/W)	1.42	4.2	1.27	4.3(1)	1.59(2)	27.3(3)
Bit Precision (bit)	N/A	8	N/A	8 (Top-1 56.9%)		1 (Top-1 56.8%)

(1) Core 10MHz, Link 15MHz, 0.7V
(2) Core 200MHz, Link 200MHz, 1.1V
(3) Core 10MHz, Link 30MHz, 0.75V

Fig. 15 illustrates a verification system used for demonstration of the UNPU and a block diagram of the verification system board. This system performs DNN acceleration by using images or texts that are transferred from mobile device as DNN inputs and transfers the DNN results to mobile devices. Dual external gateways of the UNPU are connected to external

TABLE VI
PERFORMANCE COMPARISON TABLE

	DNPU [20]		**ENVISION [19]**		**S.VLSI2017 [22]**		**BRein memory [23]**		**QUEST [25]**	**This Work**	
Purpose	CLs, RLs, FCLs		CLs		CLs, RLs, FCLs		RLs, FCLs		CLs, RLs, FCLs	CLs, RLs, FCLs	
Process [nm]	65		28		65		65		40	65	
Die Area [mm^2]	16		1.87		19.4		12		121.6	16	
Supply Voltage [V]	0.77 – 1.1		0.605 – 1.05		0.67 – 1.2		0.55 – 1.0		1.1	0.63 – 1.1	
Maximum Frequency MHz]	200		200		200		400		330	200	
PE Bit-precision [Bit]	4, 8, 16		4, 8, 16		8, 16		1, 2		1 – 4	1 – 16	
Peak Performance [GOPS]	1200 (4b) 300 (16b)		408 (4b) 76 (16b)		409.6 (8b)		1,380 (1b)		7490 (1b) 1960 (4b)	7372 (1b), 1382(4b), 691.2(8b), 345.6(16b)	
Power Consumption [mW]	50MHz 0.77V	34.6	50MHz 0.63V	7.5	10MHz 0.67V	4	100MHz 0.55V	50	3300	5MHz 0.63V	3.2
	200MHz 1.1V	279	200MHz 1.05V	300	200MHz 1.2V	447	400MHz 1.0V	600		200MHz 1.1V	297
Power Efficiency [TOPS/W]	50MHz 0.77V	8.1 (4b)	50MHz 0.63V	10.0 (4b)	10MHz 0.67V	5.09 (8b)	100MHz 0.55V	6.0 (1bx1b)	2.27 (1b) 0.59 (4b)	3.08 (16b) 11.6 (4b) 50.6 (1b)	
		2.1 (16b)	100MHz 0.7V	0.53 (16b)	200 1.2V	1.06 (8b)	400MHz 1.0V	2.3 (1bx1b)			

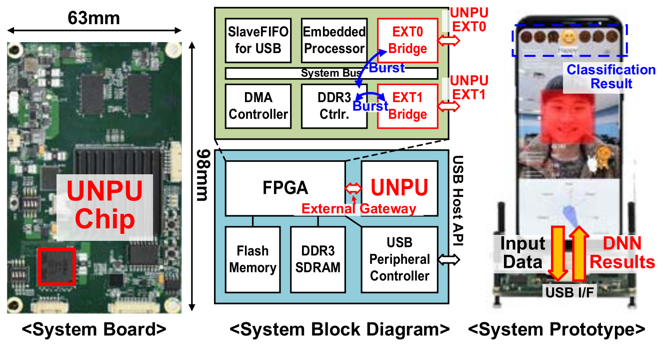


Fig. 15. Verification system and system block diagram.

bridges in the FPGA (EXT0 bridge and EXT1 bridge). These two external bridges are connected to the DDR3 memory controller in the FPGA via the system bus, allowing the UNPU to perform burst transactions with DDR3 SDRAM on the system board.

The UNPU communicates with the mobile device through the USB interface and the USB host API supported from the Android OS. The Slave FIFO in the FPGA and the USB peripheral controller chip on the system board are used to connect the UNPU to the mobile device via the FPGA. The UNPU integrated on the system board can accelerate the VGG-16 inference with the throughput of 35.6, 14.5, and 7.6 fps for the 1-, 4-, and 8-bit weight precision, respectively.

Table VI explains the performance comparison table with deep learning hardware accelerators reported. This work is the first DNN accelerator ASIC that can support all weight precision from 1 to 16 bit, unlike the other works which only support limited bit precision such as 4, 8, or 16 bit [19], [20] or 1, 2, 3, 4 bit [25]. In addition, compared with [20] designed with heterogeneous core architecture, unified DNN core architecture of this work makes the UNPU get 1.15× higher performance than the [20]. This work has a wider operating range of from 0.63 to 1.1 V compared to [20]. Thus, the UNPU has a 1.43× and 1.46× higher power efficiency than [20] for 4 and 16 bit operations, respectively.

VII. DISCUSSION

A. Bit Precision Variability for Activations (Feature Map)

There are two kinds of operands in DNN, activations and weights. Among these, the variability of weight precision is the main subject in this paper. However, there are many cases that DNNs show sufficient accuracy with low weight precision as well as low activation precision. In these cases, UNPU reduces energy and power consumption by put the low bit precision activations into AFL that is input data buffer of LBPEs. By doing this, in the VGG-16 benchmark, the power consumption of UNPU is reduced by 13%, 25%, 31%, and 35% for 8, 4, 2, and 1 bit activations, respectively, compared to 16 bit activations.

The reason why bit-serial operation is not used for activations in UNPU unlike the case of weights is as follows. For every DNN layer, there are two kinds of activations; input activations (input features) and output activations (output features). Among these, output activations are calculated by repeated accumulation of partial sums, thus, minimum bit precision to represent output activations varies during the DNN operation. In addition, data distribution of output activations is unknown until it is calculated. Therefore, the precision variability for input activations and output activations is separate issue in hardware design. In other words, even if the precision of input activation is fully variable, the precision variability of output activation remains unsolved. In addition, in order to implement fully variable precision for both of input activations and output activations, accumulation paths for every bit precision, including adder trees and partial-sum registers, must be implemented separately. It makes huge area and power overhead, and it is not desirable. Therefore, fully variable bit precision for both of input and output activations is difficult and inefficient. Instead, power efficiency can be improved by optimizing bit precision of accumulation path and by giving the input data buffers low precision activations like with UNPU.

B. Power Consumption Including Off-Chip Memory

The power consumption of DDR3 SDRAM memory controller (32 bit) estimated by intel PowerPlay power analyzer tool is 0.924 W and 2 Gb 800-MHz DDR3 SDRAM is used in prototype system. This number is over three times larger than power consumption of UNPU. Furthermore, if the DDR3 SDRAM in system board is included for power estimation, UNPU occupies only 18% power consumption among the overall memory system. In other words, the power consumption of memory system occupies more than the computational part of the overall system. Therefore, the effort to reduce power consumption of memory system including memory controller is very important to enhance power efficiency of overall system. A straightforward way to reduce memory power consumption is to lower dynamic power consumption by lowering the activity factor of memory system. This will be accomplished by reducing off-chip memory accesses of computational part in system. Some of the possible solutions to this are the integration of large-scale on-chip memory or near-memory processing.

VIII. Conclusion

The UNPU, an energy-efficient DNN inference accelerator, is proposed to realize embedded deep learning applications on mobile platforms. The UNPU supports fully variable weight bit precision from 1 to 16 bit to accelerate DNNs on the energy-accuracy optimal point, and it has unified DNN core architecture which makes the UNPU get 1.15× higher computation performance than that of [20]. The UNPU integrates LBPEs to increase energy efficiency using table-based MAC operation and it shows 53.6% lower energy consumption than the previous fixed-point MAC arithmetic units for 1-bit weight precision. The AFLs integrated in the unified DNN core align input activations required for CLs operation to exploit convolutional weight reuse in CLs. The UNPU is fabricated in 65-nm CMOS process and occupies 4×4 mm^2 die area. This is the first DNN accelerator ASIC designed with fully variable bit precision and unified DNN core architecture. The functionality of the UNPU is successfully demonstrated on the verification system using VGG-16 deep CNN.

References

[1] O. Kaoru, M. S. Dao, V. Mezaris, and F. G. B. De Natale, "Deep learning for mobile multimedia: A survey," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 13, no. 3s, pp. 1–34, Jun. 2017.

[2] Y. Shuochao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. 26th Int. Conf. World Wide Web (WWW)*, Perth, WA, Australia, 2017, pp. 351–360

[3] D. Li, X. Wang, and D. Kong. (Aug. 2017). "DeepRebirth: Accelerating deep neural network execution on mobile devices." [Online]. Available: https://arxiv.org/abs/1708.04728

[4] M. Akhil, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proc. 15th Annu. Int. Conf. Mobile Syst. (MobiSys)*, New York, NY, USA, 2017, pp. 68–81.

[5] C. Qingqing, N. Balasubramanian, and A. Balasubramanian, "MobiRNN: Efficient recurrent neural network execution on mobile GPU," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl. (EMDL)*, New York, NY, USA, 2017, pp. 1–6.

[6] N. D. Lane *et al.*, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Vienna, Austria, 2016, pp. 23:1–23:12.

[7] S. I. Venieris and C.-S. Bouganis. (2017). "fpgaConvNet: A toolflow for mapping diverse convolutional neural networks on embedded FPGAs." [Online]. Available: https://arxiv.org/abs/1711.08740

[8] 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)*, Las Vegas, NV, USA, Jun. 2016, pp. 4820–4828.

[9] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. (2017). "Pruning convolutional neural networks for resource efficient inference." [Online]. Available: https://arxiv.org/abs/1611.06440

[10] A. Tulloch and Y. Jia. (2017). "High performance ultra-low-precision convolutions on mobile devices." [Online]. Available: https://arxiv.org/abs/1712.02427

[11] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. (2015). "Compression of deep convolutional neural networks for fast and low power mobile applications." [Online]. Available: https://arxiv.org/abs/1511.06530

[12] S. Bang *et al.*, "A 288 μW programmable deep-learning processor with 270 KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 250–251.

[13] P. N. Whatmough *et al.*, "A 28 nm SoC with a 1.2 GHz 568 nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 242–243.

[14] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2016, pp. 1–2.

[15] K. Bong, S. Choi, C. Kim, D. Han, and H.-J. Yoo, "A low-power convolutional neural network face recognition processor and a CIS integrated with always-on face detector," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 115–123, Jan. 2018.

[16] 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.

[17] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "A 1.42 Tops/W deep convolutional neural network recognition processor for intelligent IoE systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2016, pp. 264–265.

[18] G. Desoli *et al.*, "A 2.9 TOPS/W deep convolutional neural network SoC in FD-SOI 28 nm for intelligent embedded systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 238–239.

[19] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–257.

[20] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.

[21] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 1–12, 2017.

[22] S. Yin *et al.*, "A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2017, pp. C26–C27.

[23] K. Ando *et al.*, "BRein memory: A 13-layer 4.2 K neuron/0.8 M synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm CMOS," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Kyoto, Japan, Jun. 2017, pp. C24–C25.

[24] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1 b-to-16 b fully-variable weight bit-precision," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 218–220.

[25] K. Ueyoshi *et al.*, "QUEST: A 7.49 TOPS multi-purpose log-quantized DNN inference engine stacked on 96 MB 3D SRAM using inductive-coupling technology in 40 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 216–218.

[26] L. Lai, N. Suda, and V. Chandra. (2017). "Deep convolutional neural network inference with floating-point weights and fixed-point activations." [Online]. Available: https://arxiv.org/abs/1703.03073

[27] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Taipei, Taiwan, Oct. 2016, pp. 1–12.

[28] C. Zhu, S. Han, H. Mao, and W. J. Dally. (2017). "Trained ternary quantization." [Online]. Available: https://arxiv.org/abs/1612.01064

[29] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. (Mar. 2016). "XNOR-Net: ImageNet classification using binary convolutional neural networks." [Online]. Available: http://arxiv.org/abs/1603.05279

[30] Q. He *et al.* (2016). "Effective quantization methods for recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1611.10176

[31] H. Sharma *et al.* (2017). "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks." [Online]. Available: https://arxiv.org/abs/1712.01507

Jinmook Lee (S'15) received the B.S. degrees in electrical engineering from Hanyang University, Seoul, South Korea, in 2014, and the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2016, where he is currently pursuing the Ph.D. degree.

His current research interests include energy-efficient deep learning inference/training accelerator ASIC design, embedded deep learning platform design and verification, embedded system development with FPGA programming, and deep learning algorithm for sequence recognition.

Changhyeon Kim (S'16) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include low-power system-on-chip design, especially focused on parallel processor for artificial intelligence and machine learning algorithms.

Sanghoon Kang (S'16) received the B.S. and M.S. degrees with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2016. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering.

His current research interests include low-power vision system-on-chip design and deep learning processor design.

Dongjoo Shin (S'13) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2013, 2015, and 2018, respectively.

His current research interests include low-power digital processors and multi-core architecture, especially in machine learning and computer vision fields.

Sangyeob Kim received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2018, where he is currently pursuing the M.S. degree.

His current research interests include low-power system-on-chip design, deep neural network accelerators, and machine learning algorithms for deep learning.

Hoi-Jun Yoo (M'95–SM'04–F'08) received the bachelor's degree from the Electronic Department, Seoul National University, Seoul, South Korea, in 1983, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1985 and 1988, respectively.

Since 1998, he has been the Faculty Member with the Department of Electrical Engineering, KAIST. From 2001 to 2005, he was the Director of the Korean System Integration and IP Authoring Research Center, Seoul. In 2007, he founded the System Design Innovation and Application Research Center, KAIST. Since 2010, he has been the General Chair for the Korean Institute of Next Generation Computing, Seoul. He is currently a Full Professor with KAIST. He has authored or co-authored the *DRAM Design* (South Korea: Hongrung, 1996), the *High-Performance DRAM* (South Korea:Sigma, 1999), the *Future Memory: FRAM* (South Korea: Sigma, 2000), the *Networks On Chips* (Morgan Kaufmann, 2006), the *Low-Power NoC for High-Performance SoC Design* (CRC Press, 2008), the *Circuits at the Nanoscale* (CRC Press, 2009), the *Embedded Memories for Nano-Scale VLSIs* (Springer, 2009), the *Mobile 3-D Graphics SoC From Algorithm to Chip* (Wiley, 2010), the *BiomedicalCMOS ICs* (Springer, 2011), the *Embedded Systems* (Wiley, 2012), and the *Ultralow-Power Short-Range Radios* (Springer, 2015). He has authored over 400 papers. His current research interests include artificial intelligence, computer vision system-on-chip (SoC), body area networks, and biomedical devices and circuits.

Dr. Yoo was a recipient of the Electronic Industrial Association of Korea Award for his contribution to DRAM technology in 1994, the Hynix Development Award in 1995, the Korea Semiconductor Industry Association Award in 2002, the Best Research of KAIST Award in 2007, the Scientist/Engineer of this month Award from the Ministry of Education, Science and Technology of Korea in 2010, the Best Scholarship Awards of KAIST in 2011, and a co-recipient of the ASP-DAC Design Award 2001, the Outstanding Design Awards of 2005–2007, 2010, 2011, and 2014 A-SSCC, and the Student Design Contest Award of 2007, 2008, 2010, and 2011 DAC/ISSCC.

He received the Order of Service Merit from the Ministry of Public Administration and Security of Korea in 2011. He has served as a member for the Executive Committee of ISSCC, the Symposium on VLSI, and A-SSCC, and the TPC Chair of the A-SSCC 2008 and ISWC 2010, the IEEE Distinguished Lecturer from 2010 to 2011, the Far East Chair for the ISSCC from 2011 to 2012, the Technology Direction Sub-Committee Chair for the ISSCC in 2013, the TPC Vice Chair for the ISSCC in 2014, and the TPC Chair for the ISSCC in 2015. From 2003 to 2005, he was the full-time Advisor to the Minister of Korea, Ministry of Information and Communication, and National Project Manager of SoC and Computer.