CNN Inference Accelerator Optimized for AI Applications for Object Detection

Bogeun Jung, Wooyoung Park, and Hyungwon Kim [a]
Department of Electronics Engineering, Chungbuk National University, Cheongju, Korea
E-mail : bogeunjung@chungbuk.ac.kr, wooyoungpark@chungbuk.ac.kr, hwkim@chungbuk.ac.kr

***Abstract* - The advancement of state-of-the-art technology has dramatically impacted the field of Deep Neural Networks (DNNs), especially Convolutional Neural Networks (CNNs). As the demand for AI applications on mobile devices grows, power-hungry GPUs are no longer viable for mobile AI applications. Instead, there is a growing research trend towards compact and low power Neural Processing Units (NPUs) to solve current problems. This article presents an efficient architecture of a CNN inference accelerator that is optimized for AI applications on mobile devices. We propose two architectural enhancements and two optimization methods to improve the existing CNN accelerator [17]. To evaluate our work, we implemented it on a Zynq UltraScale+ MPSoC ZCU 102 Evaluation Board and verified it with the YOLOv5-nano model used for CNN object detection. Experimental results show that we reduced resource utilization by 7.31% for LUTs, 22.29% for FFs, and 3.90% for DSPs. In our Vivado simulation, we accelerated the inference time by 33.5%. With the reduced resource usage described above, we have implemented an accelerator with no loss of accuracy and better resource usage and speed.**

***Keywords*—Convolutional Neural Network, CNN inference accelerator, Neural Processing Unit, YOLOv5-nano model**

I. INTRODUCTION

A Deep Neural Network (DNN) is a complex neural network with multiple layers and is one of the most popular models used in the field of Artificial Intelligence (AI). In recent years, with the advancement of AI, DNNs have received growing attention in various fields and already being utilized in various areas. Among the AI models that utilize the DNN, Convolutional Neural Network (CNN) [1], which has a structure that efficiently extracts features by considering the spatial structure of the data and identifies patterns through these features. CNNs are mainly used in object detection [2][3], classification [4][5], segmentation [6][7], and Natural Language Processing (NLP) [8][9]. While state-of-the-art DNN models employ a variation of CNN, called transformer networks, CNNs are still considered as mainstream of DNN models.

As neural networks evolve, the number of parameters increases, and the CNNs become deeper and more complex requiring faster accelerator hardware and chips. For example, ResNet-50 [10], developed by Microsoft Research, has 50 layers and 25 million parameters. Inception v4 [11], one of the CNN architectures developed by Google, has approximately 42 million parameters. AlexNet [12], a deep learning architecture for image classification that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, has eight layers and approximately 60 million parameters. VGG-16 [13], one of the CNN architectures developed by the University of Oxford that won second place in the 2014 ILSVRC competition, has 16 layers and approximately 138 million parameters. Due to this massive number of parameters, how well the hardware is optimized is becoming important.

As the demand for AI applications on mobile devices increases, power-hungry GPUs are no longer suitable for mobile AI applications. Instead, there is a growing research trend towards compact and low-power Neural Processing Units (NPUs). This work presents an efficient architecture for a CNN accelerator aimed at low-power mobile NPUs with compact on-chip memory.

The main contributions of this paper are as follows:

1) A format change for bundling activation data from the DRAM to minimize on-chip memory loss.
2) A global input buffer and global output buffer to free up on-chip memory space and flexibly manage data.
3) Minimizing the Post-Convolution Processing block to substantially reduce the size of the block and output buffer.
4) A dual buffer structure to reduce DRAM access overhead and accelerate the speed.

The rest of this paper is organized as follows: Section II introduces the overall architecture of the proposed NPU, including the first and second contributions briefly. Section III introduces the third and fourth contributions. Section IV presents the results of the RTL implementation and the analysis of the experimental results. Finally, Section V concludes this paper.

II. OVERALL ARCHITECTURE

Fig. 1 illustrates the overall architecture of the proposed NPU, which consists of a convolutional block, input buffer,

a. Corresponding author; hwkim@chungbuk.ac.kr

Manuscript Received Mar. 4, 2024, Revised Apr. 11, 2024, Accepted Apr. 13, 2024

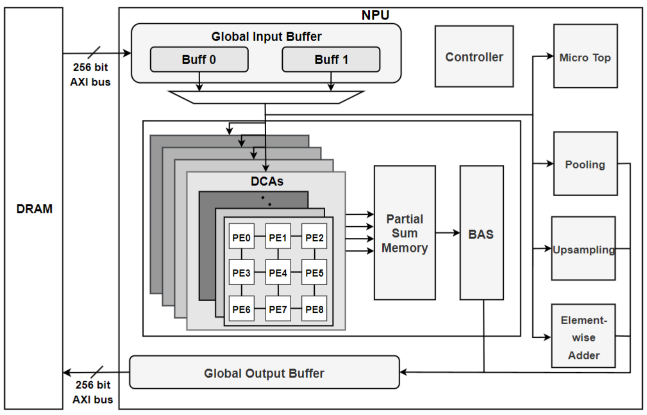
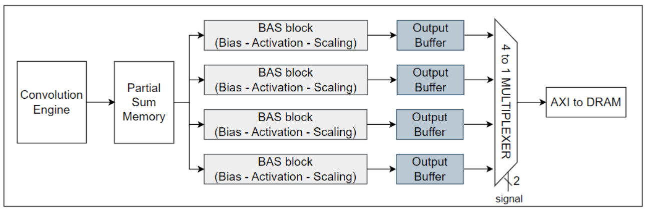


Fig. 1. Overall Architecture of proposed NPU

output buffer, microcode controller and memory, and post convolution processing blocks (Upsampling modules, Element-Wise Adder modules, and Pooling modules). The convolutional block, a central block of the proposed CNN accelerator, consists of four convolution engines called Diagonal Cyclic Arrays (DCAs), where the multiply and accumulate operations on the activation data and weights are calculated. At this point, we cannot store all the activation data and weights needed for the convolution in the on-chip memory of the NPU chip. Therefore, we employ a loop optimization method [14][15] in the proposed accelerator architecture. The loop optimization method includes loop tiling [14] and loop unrolling [14]. We also implement a loop ordering method. First, the loop tiling method divides the entire activation data into tiles of a specified size and stores them one by one in the on-chip memory before proceeding with the convolution. Loop unrolling is a way to parallelize the convolution operations using multiple tiles concurrently to reduce DRAM access and maximize data reuse. The example implementation of the proposed accelerator handles 16 input channels and 16 output channels simultaneously. Then loop ordering sets the convolution order of the tiles to be processed by the DCA blocks. The convolution order includes an input iteration that iterates by changing the input channel, a slice iteration that iterates the row and column for one tile, and an output iteration that changes the filter. In this paper, we define a tile as a slice.

Our previous CNN inference accelerator [16] fed activation data from the DRAM to the input buffer via the AXI bus in the form of 4 horizontally bundled pixels. This input buffer organization imposes limitation on the use of on-chip memory. For instance, if the data is fed in this form, it must be read in a horizontal form when processing overlapped data. At this point, the data other than the overlapped data must also be read, resulting in a loss in the utilization of on-chip memory. The improved accelerator presented in this work bundles the activation data based on the same single pixel rather than bundling four pixels horizontally for the same input channel, so that the whole input channel data for a single pixel is fed at once.

In addition, the previous NPU architecture has separate input and output buffers for the post-convolution processing blocks, Upsampling modules, Element-Wise Adder modules, and Pooling modules in addition to the buffer for the DCAs. The duplicate buffers incur unnecessarily large


Convolution Engine
Partial Sum Memory
BAS block (Bias - Activation - Scaling)
BAS block (Bias - Activation - Scaling)
BAS block (Bias - Activation - Scaling)
BAS block (Bias - Activation - Scaling)
Output Buffer
Output Buffer
Output Buffer
Output Buffer
4 to 1 MULTIPLEXER
2
signal
AXI to DRAM
Fig. 2. Previous Architecture for Post-Convolution

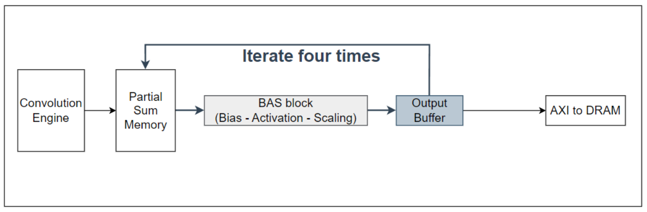


Fig. 3. Proposed Architecture for Post-Convolution

overhead on the on-chip memory. In our proposed architecture, we employ a global input buffer and a global output buffer to allow modules to share buffers, thereby reducing the on-chip memory size and making the structure of the accelerator more flexible by allowing one memory controller to manage DRAM access.

III. ARCHITECTURE ENHANCEMENT

In this section, we propose two enhancements to our previous CNN accelerator.

A. Minimizing Post-Convolution Processing block

Our previous CNN accelerator architecture includes a pipelined post-convolution processing block, called the Bias-Activation-Scaling (BAS) block, which performs post-convolution operations to produce the output features of each layer. After the activation data and weights are convolved, the partial sums for each channel are accumulated by an adder tree block and stored in the partial sum memory. These outputs are then passed to the post-convolution processing block. The bias module adds a bias parameter to the output of the convolution, and the values are passed through the Leaky Rectified Linear Unit (Leaky ReLU) in the activation module to introduce nonlinearity into the output. The scaling parameters in the scaling module then scale the output values from the activation module. Finally, the output data is stored in the output buffer and sent to the DRAM.

While the scaling process of many previous accelerator architectures converts integer to floating-point back and forth to provide higher accuracy [17][18]. Our accelerator′s scaling process uses only 16-bit pure-integer by employing an efficient architecture called Unified Scaling Pure-Integer Quantization (USPIQ) method [19]. Compared to a floating-point quantization model, this method reduced the on-chip memory for parameters by ~75% and activation data by 43.68%. Despite this reduction, we can achieve a high mAP@0.5 with a loss up to 0.61% for the YOLOv5-nano model [19].

To optimize the scaling operator in the scaling module, the output of the convolution engine is fed into BAS blocks

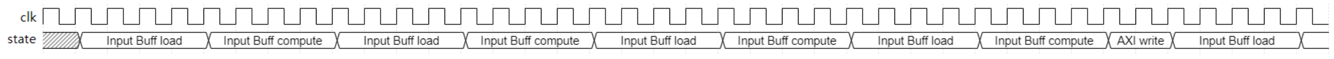


Fig. 4. Finite State Machine of previous architecture

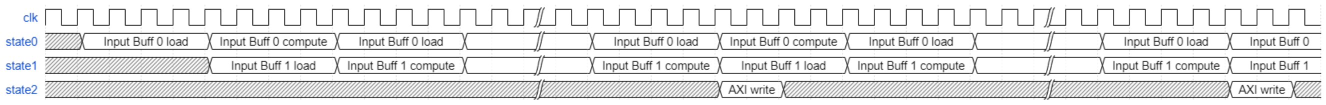


Fig. 5. Finite State Machine of proposed architecture

that can be scaled by different parameters via up to four skip connection branches. To implement this, our previous accelerator, shown in Fig. 2, had four BAS structures at the end of each partial sum memory, and multiple outputs of the convolution engine could be fed into these structures simultaneously. However, if the number of the skip connection branches for a convolution layer is more than one, the computations on the branches can be completed simultaneously, and their multiple output data should be sent from the output buffer to the DRAM via the AXI bus. When the output of one branch is sent to the DRAM, the output of the other branches must wait for it. To alleviate the above problem, the proposed accelerator shown in Fig. 3 introduces the serial BAS architecture, which processes each branch of the skip connection sequentially instead of processing multiple branches in parallel. The serial BAS block computes each branch iteratively. It can significantly reduce the size of the BAS architecture and its output buffer by a quarter. In addition, there is no additional processing time or latency compared to the previous accelerator architecture because the output data can be written to the DRAM sequentially anyway.

B. Dual buffer structure to reduce DRAM access overhead

The dual buffer (or ping-pong buffer) [20][21] is a structure used to minimize latency and optimize processing speed during data transfer. It works by alternating between two buffers to process data efficiently. This section presents the structure, operation, and performance gain of the proposed dual buffer in the example accelerator implemented targeting the YOLOv5-nano CNN model.

Our proposed accelerator operates with two input buffers. The first input buffer, Buff 0, reads weights and activation data from the DRAM for one slice size. These data are stored in the input buffer and used in the convolution operation. Once the required data are stored in Buff 0, the convolution operation is conducted using the data in Buff 0. During the convolution operation over Buff 0, the data for the next slice are read from the DRAM and stored in the second input buffer, Buff 1. Once the required data are stored in Buff 1, the convolution operation is conducted using Buff 1. By alternating between storing and processing data in this way, the dual buffer can reduce the overhead of DRAM access and improve the inference speed.

In [20], they implemented a ping-pong buffer by doubling both the input buffer and output buffer. Applying ping-pong buffer to the output buffer of the proposed architecture is inefficient. This is because there is little improvement in processing speed, considering that write access to the DRAM is relatively small and short, requiring doubling the on-chip memory in the output buffer. Therefore, we introduce a dual buffer only for the input buffer. This approach allows us to reduce the overhead of increasing on-chip memory. The rationale for this approach is that while our architecture reads duplicate data from the DRAM, the output data written to the DRAM are not duplicated.

Fig. 4 shows the partial operations of the previous architecture's Finite State Machine (FSM) and the proposed architecture's FSMs, which include states for the ping-pong buffer. Our previous architecture had only one FSM, but now it is divided into three to maximize the efficiency of the dual buffer operation, as shown in Fig. 5. Two FSMs are in charge of data loading and computing, while the remaining FSM handles the write operation of output data to the DRAM through AXI bus. By dividing the FSMs into three, the overall processing is parallelized, accelerating the speed. Additionally, in our previous architecture, the read process from the DRAM and the write process to the DRAM through AXI bus cannot be performed simultaneously, because it employs only one FSM. In contrast, in the proposed dual buffer, data are loaded and computed simultaneously, so we can separate read and write process.

A naïve design of CNN accelerators using a ping-pong buffer as the input buffer can cause errors in the operations of the FSM. This is because each layer has different loading and computation times. If one FSM has not finished loading or computing, the other FSM must wait for it. In other words, two FSMs will be in the load state at the same time if the compute time is shorter than the load time. Conversely, two FSMs will be in the compute state at the same time if the compute time is longer than the load time. In both cases, they will change addresses and fetch data from different addresses. To avoid this, the proposed dual buffer uses flag signals on both FSMs to indicate that each FSM has finished loading and computing. When both FSMs set the flag to the 'done' state, the 'start' signal is set to 1 and the two FSMs switch roles to continue loading and calculating data.

IV. RESULT AND ANALYSIS

Fig. 6 shows the entire development environment. We synthesized and implemented our CNN inference accelerator using Xilinx Vivado and Verilog HDL. Fig. 7 shows the overall layout with the RTL implementation from Vivado to the target board. Our architecture's target FPGA is Zynq UltraScale+ MPSoC ZCU102 Evaluation Board. We verified our design using a CON-FMC connector to connect the host PC and the ZCU102 Evaluation Board. As a target model, we used the YOLOv5-nano model, one of the algorithms for object detection based on deep learning, and COCO dataset.

Table I shows the resource usage comparison between the previous architecture and the proposed architecture. The LUT

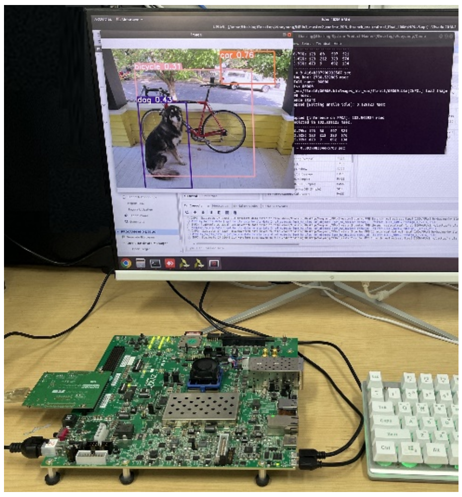


Fig. 6. Entire development environment

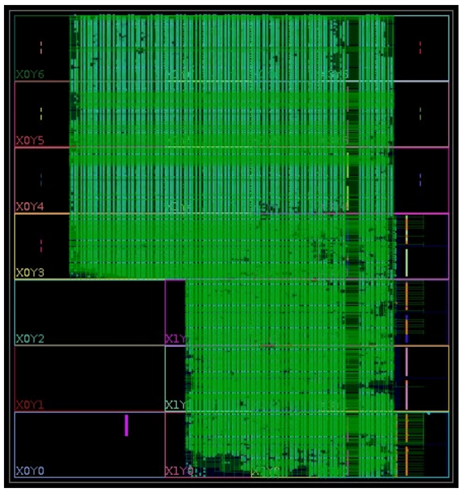


Fig. 7. Overall board layout

count has been reduced by 7.31%, Flip-Flop (FF) count has been reduced by 22.29%, BRAM remains the same size, and the DSP count has been reduced by 3.90%. The reason the size of the BRAM remains the same is that it has been reduced by 48 by applying Section IV-A method but increased again by 48 by applying Section IV-B method.

Table II and Table III show the inference time comparison for a slice size 40 and 20, respectively. According to Table II, for a slice size 40, the inference time was reduced by 43.75% in the case of 3x3 kernel stride 2 case, by 28.10% in the case of 1x1 kernel stride 1 case, and by 24.62% in the case of 3x3 kernel stride 1 case. According to Table III, for slice size 20, the inference time was reduced by 40.08% in the case of 1x1 kernel stride 1 case and by 38.63% in the case of 3x3 kernel stride 1 case. There is no 3x3 kernel stride 2 case in the inference of the YOLOv5-nano model.

V. CONCLUSION

This paper proposes two optimizations and two enhancements to our previous CNN inference accelerator. Through these methods, we can optimize FPGA resource usage, reduce on-chip memory size and accelerate inference time. The focus of future work is optimizing the slice size of the activation data. In our current architecture, the slice size is

TABLE I. RESOURCE USAGE COMPARISON

	Previous Architecture		Proposed Architecture	
Resource	Utilization	Utilization [%]	Utilization	Utilization [%]
LUT	185,102	67.54	171,564	62.60
FF	252,610	46.08	196,305	35.81
BRAM	572	62.72	572	62.72
DSP	1,230	48.81	1,182	46.90

TABLE II. INFERENCE TIME COMPARISON FOR SLICE SIZE 40

slice size 40	Previous Architecture [cycle]	Proposed Architecture [cycle]	Decrement in time [%]
3x3 stride 2	3,333,343	1,874,684	43.75
1x1 stride 1	6,687,978	4,808,235	28.10
3x3 stride 1	902,352	680,120	24.62

TABLE III. INFERENCE TIME COMPARISON FOR SLICE SIZE 20

slice size 20	Previous Architecture [cycle]	Proposed Architecture [cycle]	Decrement in time [%]
1x1 stride 1	3,023,724	1,811,685	40.08
3x3 stride 1	228,334	140,124	38.63

limited to a square shape. In Section II we introduced our optimization method for how activation data is bundled. With this method, we aim to use a slice size with a rectangular shape, where the height is longer than the width. This method allows the accelerator to increase the burst length of the AXI bus while maximizing the use of the entire on-chip memory.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A5A8026986) and supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2020-0-01304, Development of Self-learnable Mobile Recursive Neural Network Processor Technology). It was also supported by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2022-2020-0-01462) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation)" and supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1F1A1061314). In addition, this work was conducted during the research year of Chungbuk National University in 2020. The chip fabrication and EDA tool were supported by the IC Design Education Center (IDEC), Korea.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. https://doi.org/10.1145/3065386
[2] REDMON, Joseph, et al. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 779-788.
[3] REN, Shaoqing, et al. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 2015, 28.
[4] HUANG, Gao, et al. Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. p. 4700-4708.
[5] IANDOLA, Forrest N., et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv preprint arXiv:1602.07360, 2016.
[6] LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. p. 3431-3440.
[7] CHEN, Liang-Chieh, et al. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence, 2017, 40.4: 834-848.
[8] CHEN, Matthew C., et al. Deep learning to classify radiology free-text reports. Radiology, 2018, 286.3: 845-852.
[9] KIM, Yoon. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882, 2014.
[10] HE, Kaiming, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 770-778.
[11] SZEGEDY, Christian, et al. Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the AAAI conference on artificial intelligence. 2017.
[12] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012, 25.
[13] SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[14] Y. Ma, Y. Cao, S. Vrudhula and J. -s. Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 7, pp. 1354-1367, July 2018, doi: 10.1109/TVLSI.2018.2815603.
[15] J. Li et al., "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2018, pp. 343-348, doi: 10.23919/DATE.2018.8342033.
[16] Dongyeong Lee, "High Speed CNN Accelerator SoC Design Based on Scalable Array." Master's thesis in Korea, Chungbuk National University, 2023. Chungcheongbuk-do.
[17] WU, Chen, et al. Phoenix: A low-precision floating-point quantization oriented architecture for convolutional neural networks. *arXiv preprint arXiv:2003.02628*, 2020.
[18] CHOI, Dahun; KIM, Hyun. Hardware-friendly log-scale quantization for CNNs with activation functions containing negative values. In: *2021 18th International SoC Design Conference (ISOCC)*. IEEE, 2021. p. 415-416.
[19] Al-Hamid, A.A.; Kim, H. Unified Scaling-Based Pure-Integer Quantization for Low-Power Accelerator of Complex CNNs. *Electronics* 2023, *12*, 2660. https://doi.org/10.3390/electronics12122660.
[20] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15). Association for Computing Machinery, New York, NY, USA, 161–170. https://doi.org/10.1145/2684746.2689060.
[21] Y. Ma, Y. Cao, S. Vrudhula and J. -S. Seo, "Performance Modeling for CNN Inference Accelerators on FPGA," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 4, pp. 843-856, April 2020, doi: 10.1109/TCAD.2019.2897634.

Bogeun Jung received his B.S. degree in electronics engineering from Chungbuk National University, Cheongju, South Korea, in 2024. He is currently a Master's student in the MSIS lab at Chungbuk National University.

His research interests include neural network processor SoC design, CNN optimization, and AI Accelerators.

Wooyoung Park is currently a senior in the Department of electronics engineering at Chungbuk National University, Cheongju, Korea. He is an integrated Bachelor's and Master's course student in the MSIS lab at Chungbuk National University.

His research interests include neural network processor SoC design, CNN optimization, and AI Accelerators.

Hyungwon Kim received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology in 1991 and 1993, respectively, and the Ph.D. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, MI, USA, in 1999. In 1999, he joined Synopsys Inc., Mountain View, CA, USA, where he developed electronic design automation software. In 2001, he joined Broadcom Inc., San Jose, CA, USA, where he developed various network chips, including a WiFi gateway router chip, a network processor for 3G, and 10 gigabit ethernet chips. In 2005, he founded Xronet, Inc., a Korea-based wireless chip maker, where he managed the company as CEO to successfully develop and commercialize wireless baseband and RF chips and software, including WiMAX chips supporting IEEE802.16e and WiFi chips supporting IEEE802.11a/b/g/n. Since 2013, he has been with Chungbuk National University, Cheongju, South Korea, where he is currently an Associate Professor with the Department of Electronics Engineering. His current research focuses on neural network processor SoCs, CNN optimization, object recognition for autonomous driving, V2X network and security, sensor read-out circuits, touch screen controller SoCs, and wireless sensor networks.