IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 July 2022, accepted 5 August 2022, date of publication 10 August 2022, date of current version 16 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3197906

RESEARCH ARTICLE

A Low-Cost Fully Integer-Based CNN Accelerator on FPGA for Real-Time Traffic Sign Recognition

JAEMYUNG KIM[1], (Graduate Student Member, IEEE),
JIN-KU KANG[1], (Senior Member, IEEE),
AND YONGWOO KIM[2], (Member, IEEE)
[1]Department of Electrical and Computer Engineering, Inha University, Incheon 22212, South Korea
[2]Department of System Semiconductor Engineering, Sangmyung University, Cheonan 31066, South Korea

Corresponding authors: Yongwoo Kim (yongwoo.kim@smu.ac.kr) and Jin-Ku Kang (jkang@inha.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT, Ministry of Science and ICT) (No. 2022R1G1A1007415). This research was supported by the MSIT, Korea, under the Information Technology Research Center (ITRC) support program (IITP-2021-0-02052) supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP). This research was the results of a study on the High Performance Computing (HPC) Project, supported by the MSIT and National IT Industry Promotion Agency (NIPA).

ABSTRACT Traffic sign recognition (TSR) technology allows the vehicle to recognize road signs through a camera and use it for driving. For traffic safety, TSR is one of the core technologies constituting advanced driver assistance systems (ADAS), and several researches have been studied. The advent of convolutional neural networks (CNNs) has opened up new possibilities in automotive environments, especially for ADAS. However, deploying a real-time TSR application in resource-constrained ADAS is challenging because most CNNs require high computing resources and memory usage. To address this problem, some works have been studied to consider optimization in embedded platforms, but existing works used many hardware resources or showed low computation performance. In this paper, we propose a low-cost CNN-based real-time TSR hardware accelerator. Firstly, we extend a novel hardware-friendly quantization method to reduce computational complexity. The quantization method can reconstruct the CNN so that all operations, including the skip connection path of residual blocks, use only integer arithmetic and reduce the computational overhead by replacing the quantization affine mapping process with a shift operation. Secondly, the proposed hardware accelerator applied two parallelization strategies to balance real-time inference and resource consumption. In addition, we present a simple and effective hardware design scheme that handles the skip connection path of residual blocks. This design scheme can optimize the dataflow of the skip connection path and reduce additional internal memory usage. Experimental results show that the reconstructed fully integer-based CNN only requires 24M integer operations (IOPs) and possesses a model size of 0.17MB. Compared with the previous work, the proposed CNN model size was reduced by $\times 105$, and the number of operations was reduced by $\times 58$. In addition, the proposed CNN can achieve a TSR accuracy of 99.07%, which is the highest accuracy among CNN-based TSR works implemented on embedded platforms. The proposed hardware accelerator achieves a computation performance of 960 MOPS and a frame rate of 40 FPS when implemented on a Xilinx ZC706 SoC. Consequently, this work improves by $\times 11.87$ and $\times 36.7$ on computation performance and frame rate compared to the previous work.

INDEX TERMS Traffic sign recognition, CNN, quantization, accelerator, FPGA.

I. INTRODUCTION
Recently, interest in advanced driver assistance systems (ADAS) has gradually increased. Computer vision technology is being applied to various ADAS fields such as pedestrian detection, lane recognition, and drowsy driver detection. Among the technologies related to the advancement of vehicles, traffic sign recognition (TSR) is the core technology constituting ADAS. Accordingly, the demand for TSR is increasing, and many studies have been conducted.

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

Convolutional neural networks (CNNs) have been applied to various image processing tasks and showed excellent performance. In the field of TSR, some works using CNN have been devised [1], [2], [3], [4]. However, these works focus only on improving performance in GPU environments that require high computational and memory requirements. Therefore, it is unsuitable for implementation in resource-constrained automotive environments.

To address these problems, several studies considering embedded environments have been reported [5], [6], [7], [8], [9], [10]. Wong *et al.* [5] introduced an integrated design strategy and optimization method at the micro-architecture and macro-architecture levels to design CNNs with optimized information density while meeting the embedded requirements. Novac *et al.* [6] proposed a new framework for end-to-end training, quantization, and deployment of CNNs. The proposed framework quantizes and deploys CNNs onto microcontrollers. Yao *et al.* [7] suggested a method to increase CNN's parameter utilization and adopted four optimization strategies to create a CNN topology suitable for resource-constrained FPGA platforms. Maraoui *et al.* [8] implemented a hardware accelerator using the high-level synthesis (HLS) and the Xilinx PYNQ platform using a customized overlay to build a fast and powerful embedded application. Experimental results prove the effectiveness of the prototype hardware in terms of execution time and cost. Lechner *et al.* [9] proposed a method to reduce the computation cost in the forward pass by implementing efficient hardware acceleration using binary weights instead of complex computations such as batch normalization and exponential linear units. Liu *et al.* [10] reasonably divided the functional modules of each part of the network and optimized the loop design using the HLS to improve throughput and recognition speed. Some works [7], [8], [10] were implemented in a floating-point format that requires many hardware resources. The other work [6] showed slow inference speed, and the prior work [9] exhibited low recognition accuracy. Some studies [9], [10] used only ten classes for TSR work. In summary, the existing CNN-based TSR works implemented on embedded platforms are not optimized for automotive environments that require real-time inference and have constrained resources.

In this paper, we propose a low-cost fully integer-based CNN accelerator for real-time TSR. The key contributions of this paper are as follows:

- We extended a novel hardware-friendly quantization method [14] and applied it to residual blocks (RBs). The hardware-friendly quantization method can reconstruct the CNN so that all operations, including a skip connection path of residual blocks, use only integer arithmetic. Furthermore, it can replace all operations required in the quantization affine mapping process with a shift operation.
- We propose a low-cost fully integer-based CNN accelerator for real-time TSR in automotive environments. The proposed hardware accelerator is applied two parallelization strategies (kernel-wise and output channel parallelization) to balance real-time inference and resource consumption. Furthermore, we present a simple and effective hardware design scheme that handles the skip connection path of residual blocks. This design scheme, namely the internal memory skip connection path, can optimize the dataflow of the skip connection path and reduce additional internal memory usage.
- The inference speed of the proposed CNN accelerator is 40 FPS, and the computation performance is 960 MOPS on Xilinx ZC706 SoC. This is the best result compared to the other CNN-based TSR works implemented on embedded platforms. In addition, we provide analysis results on resource usage, computation performance, and energy efficiency according to the number of processing elements PEs) for FP32/INT8 hardware.

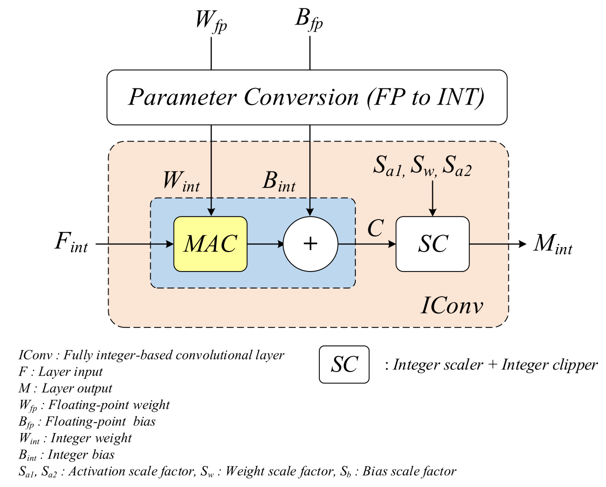

W_{fp}
B_{fp}
Parameter Conversion (FP to INT)
W_{int}
B_{int}
S_{a1}, S_w, S_{a2}
F_{int}
MAC
+
C
SC
M_{int}
IConv
IConv : Fully integer-based convolutional layer
F : Layer input
M : Layer output
W_{fp} : Floating-point weight
B_{fp} : Floating-point bias
W_{int} : Integer weight
B_{int} : Integer bias
S_{a1}, S_{a2} : Activation scale factor, S_w : Weight scale factor, S_b : Bias scale factor
SC : Integer scaler + Integer clipper

FIGURE 1. Fully integer-based convolutional layer.

The rest of this paper is organized as follows. Section II explains a hardware-friendly quantization method and fully integer-based CNN. Section III presents the proposed TSR hardware accelerator architecture. In Section IV, an analysis of the proposed accelerator and comparison results with other works are reported. Finally, we conclude this work in Section V.

II. FULLY INTEGER-BASED CNN

In this section, a hardware-friendly quantization method and fully integer-based CNN are described.

A. HARDWARE-FRIENDLY QUANTIZATION

Quantization enables efficient CNN inference by converting a high-precision floating-point domain into a low-precision fixed-point domain. Quantization can be classified into quantization-aware training (QAT) and post-training quantization (PTQ). QAT is a method of updating quantization parameters during the fine-tuning process. PTQ is a method to reduce the quantization error between floating-point and fixed-point domains by calibrating quantization parameters

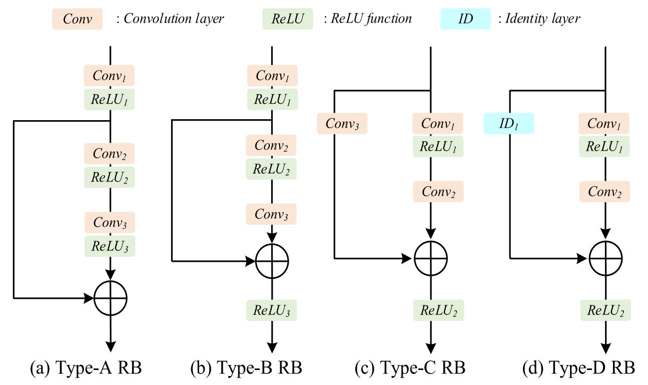


FIGURE 2. Types of residual blocks used.

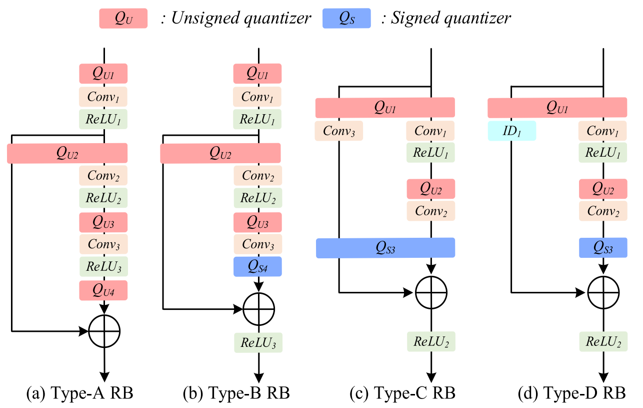


FIGURE 3. Residual blocks with quantizer.

during inference. We chose the QAT method to minimize the loss of accuracy. Two parameters (scale factor, zero point) are added for quantization. The scale factor is a parameter necessary to convert real range values into integer range values, and the zero point is a parameter required to align zeros in the two domains. Quantization parameters are trained by adding quantizers to all layer's activation, weight, and bias. The quantizer mainly consists of four sub-processes (scale, round, clip, dequantize). First, the scale process maps high-precision floating-point domain value to low-precision fixed-point domain value using the scale factor and the zero-point. The affine mapping equation needed for the scale process is as follows:

$$r = sf(q - zp) \tag{1}$$

where r is the high-precision floating-point domain value, q is the low-precision fixed-point domain value, sf is the scale factor, and zp is the zero point. Second, the round process approximates real values to the nearest integer values. The gradient discontinuity problem in the round process can be solved using a straight-through estimator (STE). Third, the clip process cuts values that exceed the bit-width set in the quantization stage. Finally, the dequantize process converts the low-precision fixed-point domain value back to the high-precision floating-point domain value using the scale factor and the zero-point. To reduce the computation overhead in the quantization affine mapping process, we introduced LLTQ [14], a novel hardware-friendly quantization method. The LLTQ method can eliminate addition operations by setting all zp values to zero. In addition, LLTQ can replace multiplication operations with shift operations by mapping sf values to power-of-two values. Therefore, the affine mapping process requires only a shift operation. The equation for directly mapping the scale factor to the power of two terms in the LLTQ method is as follows:

$$sf = \frac{2^{ceil(log_2(abs(\alpha * q\_level))}}{q\_level} \tag{2}$$

where α is a trainable parameter for quantization, and q_level is the maximum integer value that can be represented by the bit-width set during quantization (e.g., signed 8-bit data: $q\_level = 128$, unsigned 8-bit data: $q\_level = 255$).

B. FULLY INTEGER-BASED CONVOLUTIONAL LAYER

A floating-point convolutional layer (CONV) can be reconstructed as a fully integer-based CONV using the parameters obtained in the quantization-aware training (QAT) process. The reconstructed CONV is shown in Fig. 1. In Fig. 1, *SC* is a block with an integer scaler and integer clipper. The integer scaler performs shift operation using scale factors. The integer clipper adjusts the data range of the tensor according to the quantization bit-width. For example, if the quantization bit is 8-bit and the signed data has a value exceeding the range of -128 to 127, it is adjusted so that it is positioned as the last value (-128 or 127) of the data range. Since *ICONV* and *SC* are composed of one layer, the boundary between the layers becomes apparent, and the input/output becomes the same bit-width. This method enables a more uniform hardware design and allows various types of CNNs to be implemented without changing many hardware modules. The quantization-affected floating-point weight and bias parameters can be converted to integer parameters using the following equation:

$$W_{int} = clip\left(round\left(W_{fp} \ll S_w, R\right)\right) \tag{3}$$

$$B_{int} = clip\left(round\left(B_{fp} \ll S_b, R\right)\right) \gg (S_{a1} + S_w - S_b) \tag{4}$$

where S_{a1} is an activation scale factor, S_b is a bias scale factor, S_w is a weight scale factor, and R is a quantization range. Integer weight and bias parameters are converted before the inference process. The entire operation process of the reconstructed fully integer-based CONV is as follows:

$$M_{int} = clip\left(C \gg (S_{a1} + S_w - S_{a2}), R\right). \tag{5}$$

C. FULLY INTEGER-BASED RESIDUAL BLOCK

The skip connection path of RB introduced in ResNet [11] alleviates the vanishing gradient problem and shows that a deeper network can be designed. Inspired by [12], we constructed the four types of RBs shown in Fig. 2 to find the RB with the best recognition accuracy in the embedded

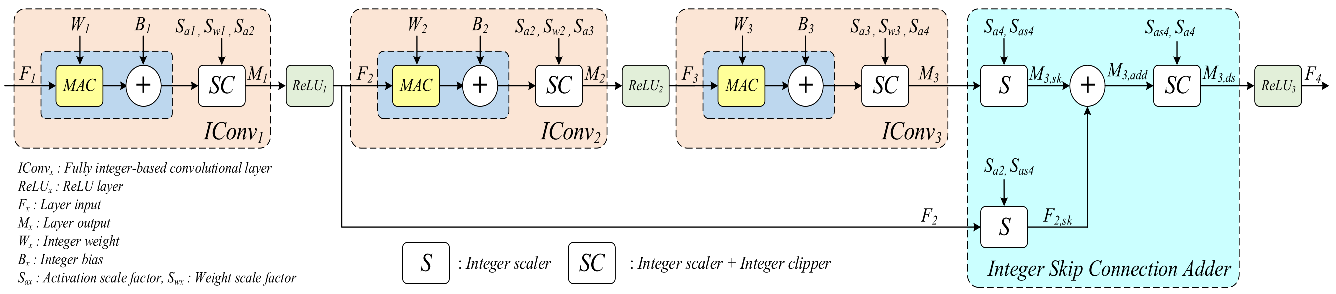


FIGURE 4. **Fully integer-based Type-B residual block.**

TSR environments. Type-A RB adds the skip path and the main path data passed through the ReLU function (ReLU). Type-B RB is a structure in which the ReLU of the main path is moved after skip connection addition from Type-A RB. Type-C RB sets the tensor size of the skip path to be the same as the main path using a CONV and then performs skip connection addition. Type-D RB is a structure in which the CONV of the skip path is changed to an identity layer (ID) [13] based on Type-C RB.

We extended the LLTQ [14] method and applied it to the four RBs. RBs, including quantizers, are shown in Fig. 3. Quantizers were inserted into the input port of the CONV and the skip connection path of each RB. In the quantization process, the type of quantizer used varies according to the input sign of the quantizer. A shared quantizer can be used if inputs from different paths have the same sign. In all four RBs in Figure 3, the starting points of skip path and main path are all unsigned values. This is because the data that has passed through the ReLU is branched. Therefore, these data can share an unsigned quantizer, and the shared quantizer is used across the second CONV and skip path. The Type-A RB uses only unsigned quantizers because all CONV inputs are unsigned. The Type-B RB uses a signed quantizer because the third CONV output of the main path is signed data. In the Type-C RB, because all data for skip connection addition are signed data, the signed quantizer can be shared and used. The signed shared quantizer is depicted behind the skip path CONV and the second CONV of the main path in Fig. 3 (c). The ID of the Type-D RB in Fig. 3 (d) does not need to add a quantizer because there is no arithmetic operation.

As will be explained later, because the Type-B RB has the best accuracy, only the integer skip connection path of the Type-B RB among the four RBs will be described in this paragraph. The fully integer-based Type-B RB is illustrated in Fig. 4. In the floating-point version, skip connection addition in RB simply adds the output of the two layers. However, for integer skip connection addition, the scale of both paths must be set equal to operate accurately. This process is called path equal scaling, and the equation for the two paths used for integer skip connection addition are described in (6) and (7):

$$F_{2,sk} = (F_2 \ll S_{a2}) \gg S_{as4} \tag{6}$$

$$M_{3,sk} = (M_3 \ll S_{a4}) \gg S_{as4} \tag{7}$$

where S_{a2} and S_{a4} are the activation scale factors of the next CONV and S_{as4} is the scale factor learned from the quantizer located at the output side of the third CONV of the main path. In the path equal scaling process, both paths ($F2$, $M3$) are adjusted to the same scale using S_{as4} after canceling the effect of the activation scale factor of the next layer. The addition of the same scaled data can be seen in (8):

$$M_{3,add} = F_{2,sk} + M_{3,sk} \tag{8}$$

After skip connection addition, down-scaling is performed according to the input scale of the next layer:

$$M_{3,ds} = clip\left(M_{3,add} \ll S_{as4}\right) \gg S_{a4}, R) \tag{9}$$

Finally, after down-scaling, the integer skip connection addition is terminated.

III. HARDWARE IMPLEMENTATION

In this section, a low-cost fully integer-based CNN accelerator architecture for real-time TSR in embedded environments is introduced.

A. ARCHITECTURE OVERVIEW

The proposed TSR accelerator is implemented on a heterogeneous SoC and is demonstrated in Fig. 5. The host CPU controls the CNN logic implemented on the FPGA using the AXI Lite interface. The AXI interface is used for data exchange between external memory and the accelerator. Input images, parameters, and output values of each layer are stored in external memory. The proposed accelerator is divided into the internal memory module (IMM) and the processing module, and the processing module is further divided into the main processing module (MPM) and the sub processing module (SPM). Each hardware component is illustrated in Fig. 6.

B. INTERNAL MEMORY MODULE

The IMM is divided into parameter memory (PRM) and pixel memory (PXM). The PRM occupies the largest size in the IMM and stores all the parameters of the layer in which the operation is conducted. The PRM is partitioned and allocated to each PE for efficient parallel MAC operations. There are two issues with implementing integer skip connection addition. First, storing skip path data in internal memory increases internal memory usage. Second, accessing

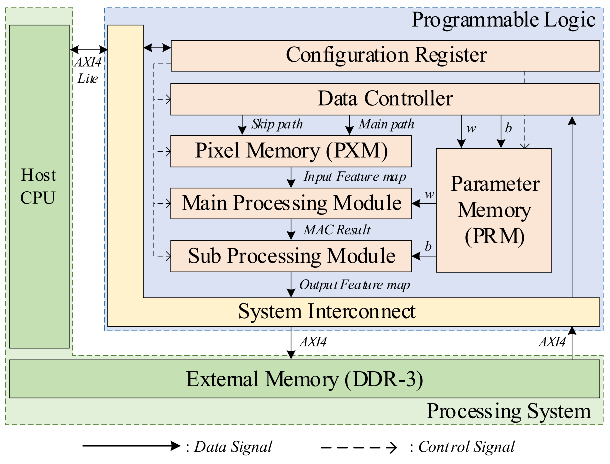


FIGURE 5. Architecture of the proposed TSR accelerator.

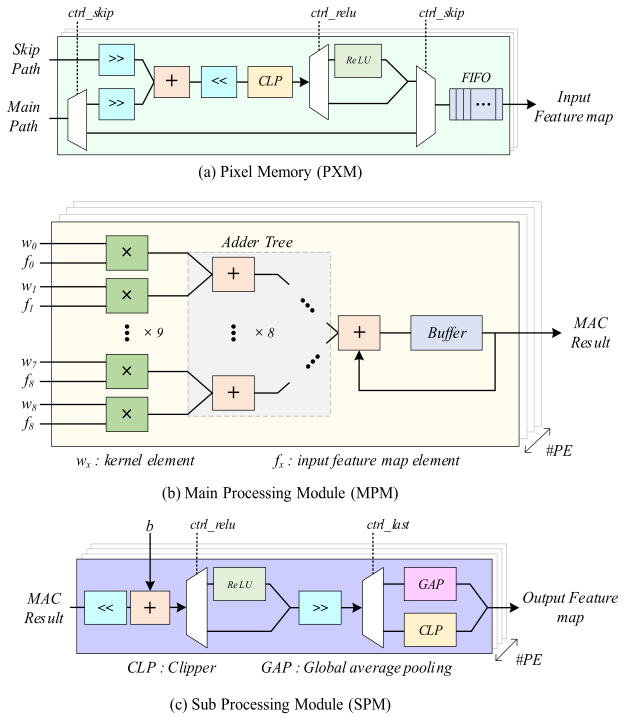


FIGURE 6. Hardware components of the proposed TSR accelerator.

the external memory to fetch the skip path data during operation in the processing module can be a bottleneck. To solve these problems, integer skip connection addition was moved from the processing module to the internal memory module. The PXM illustrated in Fig. 6 (a) is a crucial component for implementing the integer skip connection path of the RB. In our design, the skip connection adder described in Fig. 4 is implemented in PXM, not in SPM. In other words, skip connection addition is not conducted on the SPM, which is the output side of the current layer, but on the PXM, which is the input side of the next layer. When the *ctrl_skip* signal is received, both the *Skip Path* and the *Main Path* access external memory and send the results of operations (6)-(9) to the FIFO memory. Otherwise, external memory access is performed only in the *Main Path*, and this data is sent to FIFO memory. There is no need to add extra internal memories to store *Skip Path* data because the skip connection adder is implemented in PXM, not in SPM. Consequently, the internal memory skip connection path design scheme can optimize the dataflow of the skip connection path by eliminating the external memory access of the processing module and enables a more uniform hardware design.

C. PROCESSING MODULE

Fig. 6 (b) is the main processing module (MPM) that performs MAC operations in the proposed TSR accelerator. One PE consists of nine multipliers and nine adders, and the MPM is a set of PEs. In order to minimize resource usage while achieving real-time TSR, only two of the many parallelization strategies were applied to the MPM. First, we introduced a kernel-wise parallelization strategy to compute each kernel in parallel. This strategy completely unrolls the computation loop to perform the operations required for a 3×3 kernel in one cycle. As shown in Fig. 6 (b), nine multipliers and an adder tree are used for the MAC operation of one channel of the input feature map. The MAC operation result is temporarily stored in the buffer behind the adder tree. If the kernel size of CONV is 1, only one of nine multipliers of MPM operates. In addition, the MAC operation result for other channels of the input feature map is accumulated with the MAC operation result calculated in the previous cycle using the adder located right in front of the buffer. Second, we applied an output channel parallelization strategy to perform MAC operations of different filters simultaneously. This strategy partially unrolls the filter loop of CONV according to the number of PEs, so multiple PEs are implemented in the MPM. The two parallelization strategies mentioned above have no data dependency and can be implemented simply. It also uses fewer hardware resources while effectively improving computation performance than other parallelization strategy combinations, making it appropriate real-time inference for embedded environments. Fig. 6 (c) is the SPM that performs the remaining operations except for MAC operations. SPM consists of bias adder, relu function (ReLU), shifter, clipper (CLP), and global average pooling (GAP) modules. To perform parallelized post-processing synchronized with MPM, SPM was also partially unrolled according to the number of PEs. The last layer receives the *ctrl_last* signal and uses the GAP module instead of the CLP.

IV. EXPERIMENTAL RESULTS

In this section, the experimental settings and results are described. All training and evaluation processes were performed on a workstation equipped with Intel(R) Core(TM) i9-10940X CPU @ 3.30GHz, 128 GB RAM, and an NVIDIA GeForce RTX 3090 GPU. The Pytorch framework was used for floating-point training, quantization, and full integer-based inference. German traffic sign recognition

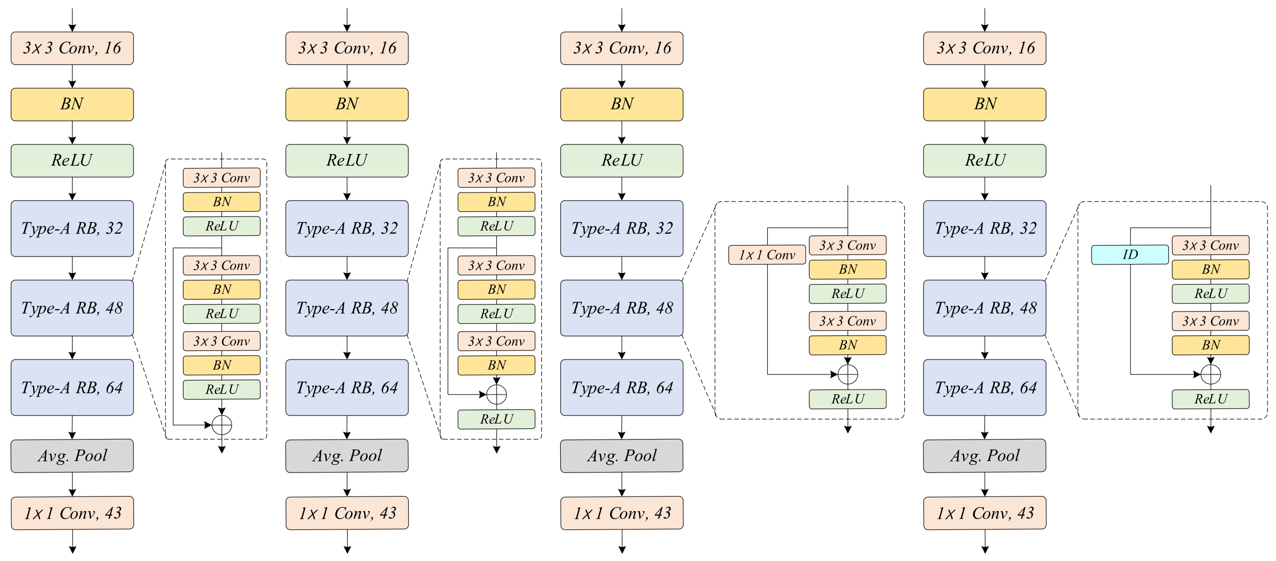


FIGURE 7. CNN architectures for residual block experiments.

benchmark (GTSRB) dataset [15] resized to 32 × 32 was used for all training and inference processes. The proposed accelerator was designed in Vivado 2019.2 tool using HLS. The Xilinx ZC706 evaluation board was used for hardware implementation and evaluation. For floating-point training, hyper-parameters are set as follows: The Adam optimizer was utilized as the training policy with the initial learning rate set to 1e-3 and the weight decay set to 1e-5. The batch size was set to 64, and the number of epochs was set to 100. For quantization-aware training (QAT), hyper-parameters are set as follows: The Adam optimizer was utilized as the training policy with the initial learning rate set to 1e-3 for quantization parameters and 1e-5 for weights and bias parameters. The batch size was set to 32, and the number of epochs was set to 50. Initialization of the quantization parameters was used with the maximum value of the first batch of the first epoch. Moreover, uniform quantizers are utilized to ensure that the low-precision fixed-point domain values are spaced uniformly, and per-tensor granularity is applied so that the quantizer learns only one quantization parameter for each tensor.

As shown in Table 1, all four CNNs repeated each RB three times in common, and by setting the stride of the first CONV of the RB to 2, the image size was reduced by half every time it passed one RB. The 1 × 1 CONV with 43 channels and GAP was used as a classifier. In the network configuration in Table 1, the superscript is the number of filters in CONV. Type-A Net, Type-B Net, and Type-C Net are composed of eleven CONVs, and the number of filters in all CONVs inside RB is the same and increases by 16 in the next RB. Since Type-D Net replaces CONV with ID, it is composed of eight CONV, and the number of filters increases by two times due to the characteristics of ID. To improve performance and convergence stability, we placed a batch

TABLE 1. The experiment results of the four types of CNN architectures for TSR.

Method	Network Configuration	Accuracy	
		FP32	INT8
Type-A Net	C^{16}-RA^{32}-RA^{48}-RA^{64}-C^{43}-GAP**	98.61*	98.46
Type-B Net	C^{16}-RB^{32}-RB^{48}-RB^{64}-C^{43}-GAP	99.14	99.07
Type-C Net	C^{16}-RC^{32}-RC^{48}-RC^{64}-C^{43}-GAP	98.32	98.38
Type-D Net	C^{16}-RD^{32}-RD^{64}-RD^{128}-C^{43}-GAP	98.13	98.09

* The best two results are highlighted in red and blue colors, respectively.
** The superscript number is the number of filters in CONV.
C : Convolutional layer, GAP : Global average pooling layer.

normalization (BN) layer between the convolutional layer and the activation function. To reduce the computational overhead and memory usage caused by BN, we applied the BN fusion method, which is fusing BN parameters with prior CONV parameters before QAT. Fig. 7 shows the detailed CNN architectures. Based on the results in Table 1, we implemented Type-B Net, which shows the best accuracy in hardware. Type-B Net passes two convolutional layers after down-sampling in the main branch of the residual block. Therefore, the information from more feature maps can be extracted, so the accuracy is higher than Type-C and Type-D Net. Type-B Net adds the ReLU function's output and the convolutional layer's output. However, Type-A Net adds two ReLU function's output. Therefore, Type-B Net can pass the information from the feature map about negative values to the next layer, resulting in higher accuracy than Type-A Net.

Table 2 shows the hardware implementation results according to the number of PEs for each precision of the proposed TSR hardware. We removed the unused PS GPIO connection and added the "*Post-Place Power Opt Design*" process in the

TABLE 2. Hardware implementation results according to the number of PEs for each precision.

Precision	FP32				INT8			
# PE	1	2	4	8	1	2	4	8
LUT	8.8K	11.5K	17.3K	28.2K	5.9K	6.2K	6.4K	7.6K
F/F	14.8K	19.5K	28.8K	49.2K	5.9K	6.2K	6.3K	7.0K
BRAM18K	61	69	82	88	18	23	25	47
DSP	54	101	195	383	13	24	40	76
PS Dynamic Power (W)	1.625	1.625	1.625	1.625	1.533	1.533	1.533	1.533
PL Dynamic Power (W)	1.121	1.431	2.187	3.182	0.225	0.234	0.257	0.441
Device Static Power (W)	0.239	0.239	0.239	0.239	0.208	0.208	0.208	0.208
Total Power (W)***	2.985	3.295	4.051	5.046	1.966	1.975	1.998	2.182
Frame Rate (FPS)	15.873	15.873	16.393	17.241	19.608	25.641****	40.000	40.000
CP (MOPS)*	380.952	380.952	393.432	413.784	470.592	615.384	960.000	960.000
EE (MOPS/W)**	127.622	115.615	97.120	82.002	239.365	311.587	480.480	439.963

* CP: Computation performance.
** EE: Energy efficiency.
*** Total on-chip power consumption is estimated using the Xilinx power analyzer.
**** The best two results are highlighted in red and blue colors, respectively.

Vivado Implementation process to optimize power consumption. Except for this, the default option of Vivado 2019.2 was used for the synthesis, and the rest of the implementation process. The power consumption was estimated using the Xilinx power analyzer. The computation performance (CP) and the energy efficiency (EE) were calculated by the following equation:

$$CP = \frac{Total\ number\ of\ operations}{execution\ time} \tag{10}$$

where the total number of operations is the total amount of computation in CNN, and the execution time is the inference time of the CNN application implemented in hardware.

$$EE = \frac{CP}{Power\ consumption}. \tag{11}$$

where the power consumption represents the total on-chip power consumption estimated using the Xilinx power analyzer.

In hardware implemented with FP32, as the number of PEs increases by two times, the resource usage increases significantly, the CP hardly changes, and the EE decreases. FP32-PE2 has no CP gain compared to FP32-PE1 because the overhead of floating-point operations is greater than the parallelization effect caused by the increase in the number of PEs. In the case of INT8 implementation, resource usage does not increase significantly even if the number of PEs increases. Moreover, the CP reaches the maximum in PE4, and the power consumption of PL increases by ×1.7 in PE8, resulting in lower EE than in PE4. INT8-PE8 has a faster computation speed because of the large number of PEs, but due to memory bandwidth limitations, the overall execution time is measured to be the same as INT8-PE4.

To sum up, the INT8-PE4 hardware shows the best CP and EE. Compared with FP32-PE4, the resource usage of INT8-PE4 is reduced ×2.7 for LUTs, ×4.6 for F/Fs, ×3.3 for BRAMs, and ×4.9 for DSPs. In addition, the CP increased by ×2.52, and the EE increased by ×4.95. Fig. 8 shows the demonstration of the proposed TSR accelerator (INT8-PE4). We measured the power consumption of the FPGA board, including all peripherals and the system fan, to provide detailed information about the total system power consumption. We applied a 12 V voltage to the FPGA board using the power supply. The multimeter is connected to the ATX power connector on the FPGA board to measure the maximum current. The maximum current when running the proposed TSR accelerator was measured to be about 0.579 A. Therefore, the total system power of the FPGA board, including all peripherals and the system fan, used in this work is 6.948 W (0.579 A × 12 V), which mainly comes from the considerable standby power of the Xilinx ZC706 evaluation board.

Although not described in Table 2, we implemented the fully integer-based Type-B Net in PS (CortexA9) using only software as a reference design. The reference design was

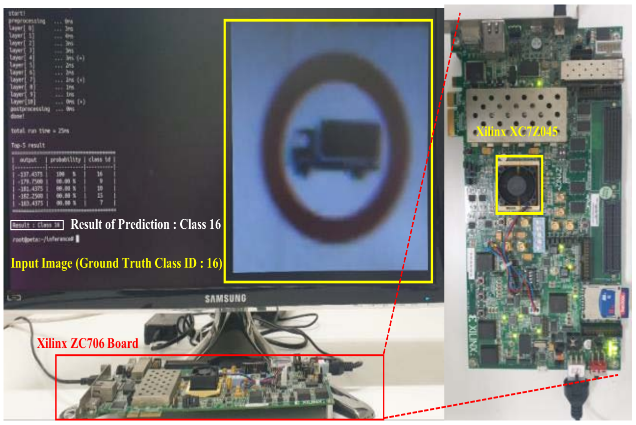

Result of Prediction : Class 16
Input Image (Ground Truth Class ID : 16)
SAMSUNG
Xilinx ZC706 Board
Xilinx XC7Z045

FIGURE 8. Demonstration of the proposed TSR accelerator.

TABLE 3. Comparison with other works.

Method	[5]	[6][*]		[7]	[8]	[9]	[10]	**This work**
Platform (CPU)	CortexA53	CortexM4F		CortexA9	CortexA9	CortexA9	CortexA9	**CortexA9**
Platform (FPGA)	N/A	N/A		XC7Z020	XC7Z020	XC7Z020	XC7Z020	**XC7Z045**
Image Size	48	32	32	N/A	32	32	32	**32**
Precision	FP16	FP32	INT8	FP32	FP32	W1A8[**]	FP32	**INT8**
Model Size (MB)	1.05	0.60	0.15	17.93	0.31	0.06	0.31	**0.17**
#OPs (M)	21	145	145	1406	9	10	4.6	**24**
#Classes	43	43	43	43	43	10	10	**43**
Accuracy (%)	98.90	97.00	97.00	98.10	N/A	96.56	98.41	**99.07**
Clock (MHz)	1200	48	48	N/A	N/A	100	100	**250**
LUT	N/A	N/A	N/A	43.2K	33.8K	30.4K	N/A	**6.4K**
F/F	N/A	N/A	N/A	36.8K	30.5K	24.7K	N/A	**6.3K**
BRAM	N/A	N/A	N/A	238	167	256	N/A	**25**
DSP	N/A	N/A	N/A	220	120	0	N/A	**40**
Frame Rate (FPS)	31.07	1.09	4.13	N/A	33.33	36.00	17.58	**40.00**
CP (MOPS)[***]	652.38	157.44	599.17	N/A	300.00	361.01	80.87	**960.00**
Power Consumption (W)	3.00	0.016	0.016	N/A	N/A	N/A	N/A	**6.95**[****]

[*] The inference time, accuracy and power consumption were used the CNN implemented with 64 filters on the Nucleo-L452RE-P platform.
[**] Weight: 1-bit, Activation: 8-bit.
[***] CP: Computation performance.
[****] This value represents the total power consumption of the FPGA board including all peripherals. The estimated total on-chip power consumption is 1.998 W.

built on the C++ compiler using the synthesizable HLS code to implement the hardware accelerator. Moreover, the reference design can be considered as INT8-PE0 (sequential application) because the parallelization pragmas used in the HLS tool, such as #pragma unroll and #pragma pipeline, are commented out in the C++ compiler. The optimization option is O3, and the operating clock of the CortexA9 processor is 667 MHz. The estimated power consumption of PS using the Xilinx power analyzer was 1.533W, and the frame rate was measured at 2.68 FPS. In addition, the reference design's CP and EE are 64.343 MOPS and 41.971 MOPS/W, respectively.

The comparison with previous works is summarized in Table 3. [5] is implemented on ARM SoC, [6] is implemented on ARM MCU, and [7], [8], [9], [10] are implemented on heterogeneous SoC. The accuracy of fully integer-based Type-B Net is about 99.07 %. The accuracy of the proposed CNN showed the highest accuracy compared to [5], [6], [7], [8], [9], [10]. This work achieved the highest accuracy because the proposed CNN did not fall into the local optimum in the floating-point training process due to BN. To summarize the proposed TSR accelerator (INT8-PE4) before comparing it with previous works, the estimated total on-chip power consumption is about 1.998W. The proportion of PS, PL, and device static in the estimated total on-chip power consumption are 76.73 %, 12.86 %, and 10.41 %, respectively. The total system power of the FPGA board is about 6.948 W. In addition, the frame rate and the CP of the proposed accelerator are 40 FPS and 960 MOPS, respectively.

Our proposed TSR hardware accelerator showed the fastest inference speed and computation performance. In addition, our hardware uses less of all hardware resources than [7] and [8]. Compared with [9], our design uses fewer hardware resources except for DSPs. A direct comparison of the proposed TSR accelerator's power consumption and energy efficiency with the previous works requires caution. This is because the same metrics between different hardware may not be suitable for direct comparison due to different platforms and evaluation methodologies. Additionally, [7], [8], [9], [10] implemented on a similar platform does not describe power consumption. Therefore, we did not directly compare power consumption and energy efficiency with other TSR hardware implemented on different embedded platforms for a fair comparison. However, we left the information on power consumption in Table 3 to provide a reference indicator for our proposed TSR accelerator.

V. CONCLUSION

In this paper, we proposed a low-cost CNN-based real-time TSR accelerator optimized for embedded environments. Our design was optimized at the software and hardware level. In software, we extended a novel hardware-friendly quantization method and applied it to residual blocks. The quantization method can reconstruct the CNN so that all operations, including the skip connection path of residual blocks, use only integer arithmetic. In addition, computational overhead can be reduced by replacing the quantization affine mapping process with a shift operation.

In hardware, two parallelization strategies were applied to minimize hardware resource consumption while achieving real-time TSR. Furthermore, a simple and effective hardware design scheme that handles the skip connection path of residual blocks is introduced to optimize the dataflow of the skip connection path and reduce additional internal memory usage. The proposed hardware accelerator is implemented on a heterogeneous SoC, and it outperforms other TSR works implemented on embedded platforms in terms of accuracy, nference speed, and computation performance. This work is fairly suitable for real-time TSR in resource-constrained automotive environments.

ACKNOWLEDGMENT

The authors thank the IDEC program and for its hardware and software assistance for the design and simulation.

REFERENCES

[1] J. Jin, K. Fu, and C. Zhang, "Traffic sign recognition with Hinge loss trained convolutional neural networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 5, pp. 1991–2000, Oct. 2014.

[2] A. Arcos-García, J. Álvarez-García, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Netw.*, vol. 99, pp. 158–165, Mar. 2018.

[3] Z. Qiumei, T. Dan, and W. Fenghua, "Improved convolutional neural network based on fast exponentially linear unit activation function," *IEEE Access*, vol. 7, pp. 151359–151367, 2019.

[4] S. Bouaafia, S. Messaoud, A. Maraoui, A. C. Ammari, L. Khriji, and M. Machhout, "Deep pre-trained models for computer vision applications: Traffic sign recognition," in *Proc. 18th Int. Multi-Conf. Syst., Signals Devices (SSD)*, Mar. 2021, pp. 23–28.

[5] A. Wong, M. J. Shafiee, and M. S. Jules, "MicronNet: A highly compact deep convolutional neural network architecture for real-time embedded traffic sign classification," *IEEE Access*, vol. 6, pp. 59803–59810, 2018.

[6] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, p. 2984, Apr. 2021.

[7] Y. Yao, Z. Zhang, Z. Yang, J. Wang, and J. Lai, "FPGA-based convolution neural network for traffic sign recognition," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, Oct. 2017, pp. 891–894.

[8] A. Maraoui, S. Messaoud, S. Bouaafia, A. C. Ammari, L. Khriji, and M. Machhout, "PYNQ FPGA hardware implementation of LeNet-5-based traffic sign recognition application," in *Proc. 18th Int. Multi-Conf. Syst., Signals Devices (SSD)*, Mar. 2021, pp. 1004–1009.

[9] M. Lechner, A. Jantsch, and S. M. P. Dinakarrao, "ResCoNN: Resource-efficient FPGA-accelerated CNN for traffic sign classification," in *Proc. 10th Int. Green Sustain. Comput. Conf. (IGSC)*, Oct. 2019, pp. 1–6.

[10] J. Liu, Z. Zhang, L. Zheng, Y. Wen, F. Bin, and L. Tang, "Traffic sign recognition based on ZYNQ," in *Proc. 9th Int. Symp. Next Gener. Electron. (ISNE)*, Jul. 2021, pp. 1–3.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Oct. 2016, pp. 630–645.

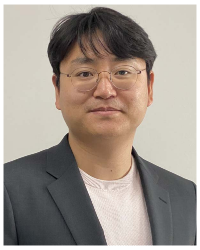
[13] Yerlan Idelbayev. *Proper ResNet Implementation for CIFAR10/CIFAR100 in PyTorch GitHub*. Accessed: Jan. 30, 2020. [Online]. Available: https://github.com/akamaster/pytorch_resnet_cifar10

[14] J. Kim, J.-K. Kang, and Y. Kim, "A resource efficient integer-arithmetic-only FPGA-based CNN accelerator for real-time facial emotion recognition," *IEEE Access*, vol. 9, pp. 104367–104381, 2021.

[15] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The German traffic sign recognition benchmark: A multi-class classification competition," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2011, pp. 1453–1460.

JAEMYUNG KIM (Graduate Student Member, IEEE) received the B.S. degree in electronics engineering from Inha University, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. His research interests include system-on-chip design, field-programmable gate array-based design, and convolutional neural network optimization and acceleration.

JIN-KU KANG (Senior Member, IEEE) received the B.S. degree from Seoul National University, Seoul, South Korea, in 1983, the M.S. degree in electrical engineering from the New Jersey Institute of Technology, Newark, NJ, USA, in 1990, and the Ph.D. degree in electrical and computer engineering from North Carolina State University, Raleigh, NC, USA, in 1996. From 1983 to 1988, he was at Samsung Electronics Inc., South Korea, where he was involved in memory design. In 1988, he was with Texas Instruments, South Korea. From 1996 to 1997, he was with Intel Corporation, Portland, OR, USA, as a Senior Design Engineer, where he was involved in high-speed I/O and timing circuits for processors. Since 1997, he has been with Inha University, Incheon, South Korea, where he is currently a Professor and leads the System IC Design Laboratory, Department of Electronics Engineering. His research interests include high-speed/low-power mixed-mode circuit design for high-speed serial interfaces.



YONGWOO KIM (Member, IEEE) received the B.S. and M.S. degrees from Inha University, Incheon, South Korea, in 2007 and 2009, and the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2019. From 2009 to 2017, he was a Senior Engineer at Silicon Works Company Ltd., Daejeon. From 2019 to 2020, he was a Senior Researcher at the Artificial Intelligence Research Division, Korea Aerospace Research Institute, Daejeon. Since 2020, he has been with Sangmyung University, Cheonan, South Korea, where he is currently an Assistant Professor with the Department of System Semiconductor Engineering. His current research interests include image/video processing algorithm, super-resolution, and deep learning hardware architecture for vision processing.