Received February 17, 2022, accepted March 16, 2022, date of publication March 24, 2022, date of current version April 1, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3162066

Energy-Efficient High-Speed ASIC Implementation of Convolutional Neural Network Using Novel Reduced Critical-Path Design

SUN SIK LEE[1], THANH DAT NGUYEN[1], PRAMOD KUMAR MEHER[2], (Senior Member, IEEE), AND SANG YOON PARK[1], (Member, IEEE)
[1]Department of Electronic Engineering, Myongji University, Yongin 17058, South Korea
[2]Department of Computer Science and Engineering, C. V. Raman Global University, Bhubaneswar, Odisha 752054, India

Corresponding author: Sang Yoon Park (sypark@mju.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) funded by the Korean Government (MSIT) under Grant 2021R1F1A1050040.

ABSTRACT Convolutional Neural Network (CNN) plays an important role in several machine learning tasks related to speech, image, and video processing applications. The increasing demand for faster processing in real-time applications requires high-speed implementation of CNN. However, in general, CNN involves higher latency due to the computationally intensive behavior of the convolutional layer. While state-of-the-art architecture provides efficient dataflow of the convolutional operations, this paper proposes a hardware-efficient, high-speed convolution block for ASIC implementation of the CNN algorithm. The proposed convolution block is designed using a novel bit-level-multiply-accumulator (BLMAC) with a modified Booth encoder and a Wallace reduction tree. The critical path of the overall architecture is significantly shortened due to the time-optimized implementation of the proposed BLMAC, which is a main component of the convolution process. Critical path analysis and dataflow strategy are also provided to demonstrate the acceleration of the proposed design. The proposed architecture was synthesized using Synopsys Design Compiler to prove its accelerated processing. The ASIC synthesis results of the proposed architecture using a 65nm standard cell library show at least 53% reduction in latency, 52.2% reduction in area-delay product, and 54.2% reduction in power-delay product compared to the state-of-the-art architecture.

INDEX TERMS Convolutional neural network (CNN), convolution layer, modified Booth encoder, Wallace reduction tree, ASIC.

I. INTRODUCTION

Convolutional neural network (CNN) has demonstrated high level of efficacy and accuracy in a variety of complex machine learning tasks, including intelligent speech and image processing, natural language processing, pattern recognition, object detection, anomaly detection, and many others [1]–[5]. Followed by AlexNet, the CNN variant proposed by Alex Krizhevsky *et al.*, which could show excellent performance in the ImageNet competition in 2015 [6], several efficient variants of CNN algorithms such as Visual Geometry Group (VGG), GoogleNet, ResNet, etc. have been proposed. While these architectures offer better performance, the computational complexity of the CNN is increased exponentially as the number of parameters is increased. Hence, the reduction in computation-time becomes a challenging task to speed up the CNN for various machine learning applications. Several efforts have been made to accelerate CNN for intelligent edge computing applications. From a hardware implementation perspective, digital signal processing (DSP) chips, graphical processing unit (GPU), field programmable gate array (FPGA), and application specific integrated circuit (ASIC) are possible platforms for the CNN implementation. Among these existing hardware platforms, ASIC, generally, requires high initial development cost, but it is the most efficient one that realizes high throughput, low latency, and small chip area with low power consumption [7].

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

The CNN is typically composed of a multiple convolutional layer followed by a pooling layer and a fully connected (FC) layer [8]. It is often required to have a large number of convolutional layers in CNN in order to increase the functional accuracy [9]. However, as the number of convolutional layers increases, so does the computational complexity and the cost of implementation. Since the computation of convolution involves more than 95% of the CNN operation [10], several efforts have been made to efficiently accelerate the convolution process. Ardakani *et al.* [11] propose an efficient architecture to realize convolutional layers of VGGNet. In addition, they coherently integrate the dataflow of the convolutional process with the computational core of the FC layer of the state-of-the-art VGG architecture. Based on their validated dataflow, this paper proposes an area-time-power efficient processing element (PE) for convolutional layer computations. The proposed architecture employs a modified Wallace reduction tree (WRT) and a modified Booth encoder (MBE) with bit-level pipelining to accelerate the processing. The contributions of this paper are outlined as follows:

1) A novel high-speed bit-level-multiply-accumulator (BLMAC) based on modified WRT and MBE is designed to reduce the latency of CNN operations.
2) A novel dual-clock strategy is proposed to improve the hardware utilization as well as overall latency where the MAC operations are accelerated with a clock with a short period while their accumulation operates with a longer clock period.
3) An area-time-power efficient hierarchical structure of the processing element with bit-level error correction is proposed.
4) An efficient dataflow strategy is proposed to efficiently utilize neurons with lower latency.
5) Precise critical path analysis is performed and critical path delay is significantly reduced to accelerate the computation of convolutional layer.
6) The effectiveness of the proposed architecture is demonstrated by the results estimated through the synthesis of proposed design.

The rest of the paper is organized as follows. Section II introduces the CNN structure and research background for several CNN architectures. The VGG16 network and its convolutional behavior are also described in this Section. The architectures of the proposed BLMAC and processing element are presented in Section III. In Section IV, the dataflow design and latency analysis of the proposed architecture are presented. Section V provides the critical path analysis to find the appropriate timing constraints for effective hardware utilization and high-speed processing of convolution operation. In Section VI, the result of ASIC synthesis of the proposed architecture is discussed and compared with the existing design. Applications of the proposed architecture are also discussed in this Section. Finally, Section VII concludes this paper.

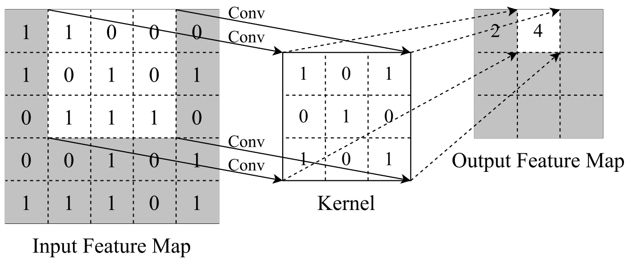


FIGURE 1. The typical functionality of a convolutional layer.

II. ALGORITHM AND ARCHITECTURE OF CNN

This section summarizes the operation of the CNN algorithm and VGGNet architecture with 16 layers, namely, VGG16. A widely used reference hardware architecture is also introduced for VGG16 implementation.

A. CNN AND VGG16 ARCHITECTURE

The CNN extracts the relevant features through a series of convolutional layers, max pooling layers, and fully-connected (FC) layers. After passing through these layers, the input is turned into a single vector, which is easy to be processed for recognition and prediction, etc. The convolutional layer is the core building block of CNN since it is used to extract feature information by carrying out the convolution operation of the output from the previous layer with the weights of the current layer. To look at the functional aspect, the convolution process applies a set of learnable filters called kernels to the input to create a feature map [12]. Fig. 1 depicts the typical convolutional layer functionality of passing an input image through the kernel to produce an output image. The input and output images of the convolution operation are called input feature map *IFMap* and output feature map *OFMap*, respectively. Considering the 3×3.2-D kernel as shown in Fig. 1, the width and height of *IFMap* are 2 pixels larger than those of *OFMap*, which takes into account the padding of the boundary.

The pooling layer is inserted after each convolutional layer, which reduces the number of parameters and speeds up by down-sampling the adjacent pixels to retrieve the optimal features. Features extracted by the convolution and pooling layers are flattened into a single vector whose element is composed of small details of the input image at high-level features. While the extracted high-level features could be connected to the output layer, a FC layer is finally used to map the extracted features into the desired outputs.

VGG16 is a CNN architecture proposed by Simonyan and Zisserman in 2015 [13]. At the ILSVRC 2014 competition, VGG16 showed outstanding results, taking second place in the overall event. The model achieved 92.7% top-5 test accuracy on ImageNet, a dataset of over 14 million images belonging to 1000 classes. The VGG16 network architecture is summarized in Table 1 [14]. The VGG16 architecture contains 13 convolutional layers grouped into five convolution sets. Each set uses 3×3 convolution filters

TABLE 1. VGG16 network architecture for the classification of RGB image of size 224 × 224 for 1000 classes.

Operation	Input Image Size	Output Image Size
Conv(3×3×3, 64)	224×224×3	224×224×64
Conv(3×3×64, 64)	224×224×64	224×224×64
max Pooling(2, 2)	224×224×64	112×112×64
Conv(3×3×64, 128)	112×112×64	112×112×128
Conv(3×3×128, 128)	112×112×128	112×112×128
max Pooling(2, 2)	112×112×128	56 ×56 ×128
Conv(3×3×128, 256)	56 ×56 ×128	56 ×56 ×256
Conv(3×3×256, 256)	56 ×56 ×256	56 ×56 ×256
Conv(3×3×256, 256)	56 ×56 ×256	56 ×56 ×256
max Pooling(2, 2)	56 ×56 ×256	28 ×28 ×256
Conv(3×3×256, 512)	28 ×28 ×256	28 ×28 ×512
Conv(3×3×512, 512)	28 ×28 ×512	28 ×28 ×512
Conv(3×3×512, 512)	28 ×28 ×512	28 ×28 ×512
max Pooling(2, 2)	28 ×28 ×512	14 ×14 ×512
Conv(3×3×512, 512)	14 ×14 ×512	14 ×14 ×512
Conv(3×3×512, 512)	14 ×14 ×512	14 ×14 ×512
Conv(3×3×512, 512)	14 ×14 ×512	14 ×14 ×512
max Pooling(2, 2)	14 ×14 ×512	7 ×7 ×512
Fully-Connected	25,088	4,096
Fully-Connected	4,096	4,096
Fully-Connected	4,096	1,000
Softmax		

across the entire network. In Table 1, Conv($3 \times 3 \times m$, N) denotes the convolution of the input image with N $3 \times 3 \times m$ kernels to create N output images. After each convolution set, a 2×2 max pooling is performed to reduce the output image size. Three FC layers are followed at the end of the last pooling layer, and the final Softmax function converts the values to indicate their relative importance.

As the study of [15], Quan Liu *et al.* demonstrated that the VGG16 has the fastest convergence behaviour, the shortest training time, and the highest functional accuracy in CNN performance. Based on this comparative study, the VGG16 is recommended as one of the most suitable models for the realization of CNN.

B. CONVOLUTION OPERATION

As the name of the algorithm implies, the convolutional layer is the most important and computation-intensive module in VGG16 [16]. The convolution operation extracts features of the image by multiplying each element of the kernel to the input image and adding the results together. The kernel moves through the whole *IFMap* by a suitable predetermined stride. To avoid loss of boundary information, the VGG16 architecture pads zeros around the boundary of the *IFMap*.

Algorithm 1 shows a pseudo code of a 2-dimensional operation in the convolutional layer, where element-wise multiply-accumulate (MAC) operation between the *IFMap*

Algorithm 1 Pseudo Code for 2-D Operation in Convolution Layer

```
input  : 2-D IFMap $I(W_I, H_I)$,
         Kernel $W(W_W, H_W)$,
         Stride $S$
output: 2-D OFMap $O(W_O, H_O)$
$W_O = (W_I - W_W)/S + 1$
$H_O = (H_I - H_W)/S + 1$
for $i \leftarrow 0$ to $H_O - 1$ do
    for $j \leftarrow 0$ to $W_O - 1$ do
        for $k \leftarrow 0$ to $H_W - 1$ do
            for $l \leftarrow 0$ to $W_W - 1$ do
                $O_{i,j} += I_{i \times S+k, j \times S+l} \times W_{k,l}$
```

and the kernel is depicted. Note that the 2-D kernel size and stride in VGGNet are fixed to 3×3 and one pixel for the convolutional process, respectively.

As shown in Algorithm 1, CNN is basically based on MAC computation. Therefore, from a hardware implementation point of view, the optimization of MAC has a strong potential to speed up the entire CNN processing. Many studies have proposed efficient hardware implementations based on various CNN structures to achieve speed improvement [11], [17]–[20]. This paper focuses on optimizing the MAC computation of CNN and the PEs that contain the MAC unit. Therefore, a reference architecture of CNN is needed to verify the proposed MAC and the PE architecture. Arash Ardakani *et al.* [11] have proposed a novel computation of the convolutional layer using VGGNet with a 2-D kernel size of 3×3, and demonstrated better performance than other approaches [17]–[20]. Therefore, we have used the structure of [11] as a reference architecture to verify the performance of the proposed PE. However, since the CNN algorithms are basically based on MAC computations, the proposed PE can also be applied to other types of CNN implementations.

III. ARCHITECTURE OF PROPOSED PROCESSING ELEMENT

A. REFERENCE ARCHITECTURE OF THE CONVOLUTION ENGINE

As can be seen in the literature, CNN hardware accelerators are of two basic types, e.g., (i) the fine-grained structure and (ii) the coarse-grained structure [21]. While fine-grained implementation uses a large number of small PEs, the coarse-grained architecture uses fewer PEs having more computing power. In this work, we propose a hierarchical architecture that can improve the overall performance, taking into account coarse-grained PEs with finer-granularity of operation within the MAC.

Fig. 2 shows the reference architecture of a convolution engine (CE) using the proposed PE architecture. The CE consists of a weight generator and three PEs for 3×3 convolution, where each PE performs the MAC computations

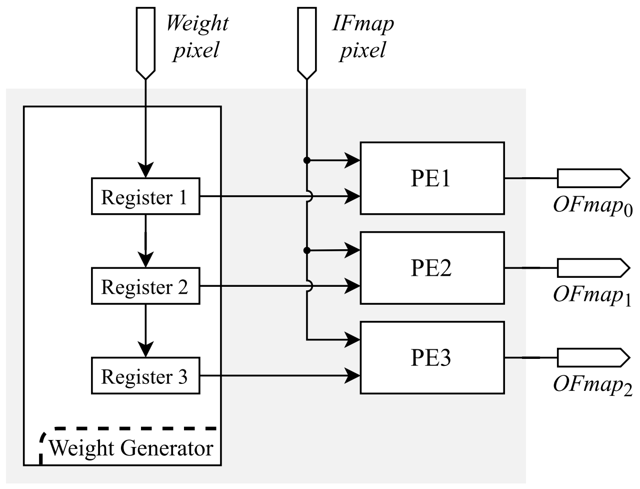


FIGURE 2. A reference architecture of the convolution engines (CE).

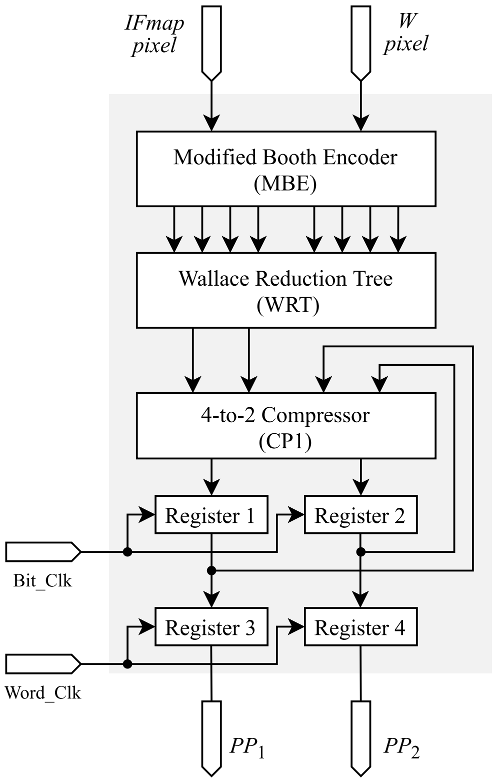


FIGURE 3. Architecture of the proposed BLMAC.

within one kernel window. Each of the PEs receives an input pixel and its weight from the weight generator. The dataflow indicating the order in which the input pixels and weights are fed in each clock cycle is described in Section IV.

B. ARCHITECTURE OF PROPOSED BIT-LEVEL MAC

Since the MAC operations comprise the core computation of the PEs, in order to overcome the low throughput drawback of the existing PE architectures, we propose here a speed-optimized bit-level MAC, namely BLMAC. The MAC unit generally performs multiplication followed by successive accumulation. In an integrated form, both the operations can be combined by realizing it through three different sections: (i) partial product (PP) generation, (ii) partial product addition, and (iii) output accumulation. We present here a novel architecture of MAC unit which is the core computing unit of the proposed PE as shown in Fig. 3. The MAC unit multiplies the pixel values in each row of the *IFMap* by a weight value, and then adds the product value to the accumulated result.

Specifically, the Booth encoder can be used to generate the partial products (PPs) corresponding to the multiplication of an input pixel with a kernel value. The modified Booth encoder (MBE) proposed in [22] is used to reduce the number of PPs in half, i.e., $m/2$ instead of m, for m-bit multiplication. The performance of the CNN algorithm converges in 16-bit resolution according to [16], thus in the proposed BLMAC, m is set to 16. The pseudo code for generating eight PPs is given in Algorithm 2. Also, Fig. 4 shows the bits of each of the eight PPs generated by the 16-bit MBE.

The product value is obtained by adding up all the eight PPs produced by the MBE. The simplest way to perform this is to use an adder-tree composed of seven ripple carry adders (RCAs). However, the propagation delay of k-bit RCA increases proportionally with k since

$$T_{\mathrm{RCA}} = k \times T_{\mathrm{FA}}, \tag{1}$$

where T_{FA} is the propagation delay of a full adder (FA). Therefore, the delay of multiplication increases proportionally with the number of PPs. In order to accelerate processing in the proposed architecture, the Wallace reduction tree (WRT) is used to reduce the $m/2$ PPs to two PPs [23]. The main task of WRT is to group two or three bits at the same bit position and use a half adder (HA) or a full adder (FA) to reduce to two bits over two consecutive bit positions. This process continues until sum of the PPs are reduced to two words by the WRT. The dot diagram of WRT for reducing eight PPs generated by 16-bit MBE to two words over four-stages is shown in Fig. 5(a).

The proposed BLMAC architecture does not employ RCA to add the final two words generated by WRT, to avoid a long propagation delay of RCA as required according to (1). In the proposed structure, these two words are stored in two separate registers (Register 1 and Register 2 as shown in Fig. 3). In the next clock cycle, two other words, corresponding to another multiplication, are generated by the WRT, and added to the previous result stored in Register 1 and Register 2. The process of reducing the four words to two is carried out by a 4-to-2 compressor denoted as CP1 (as shown in Fig. 3). The detailed dot diagram of the compressor is shown in Fig. 5(b). Considering 3 × 3 kernel filter, this process is performed over three clock cycles. The accumulated results are passed to other two registers marked as Registers 3 and 4 while Registers 1 and 2 are reset to receive the partial results corresponding to the next

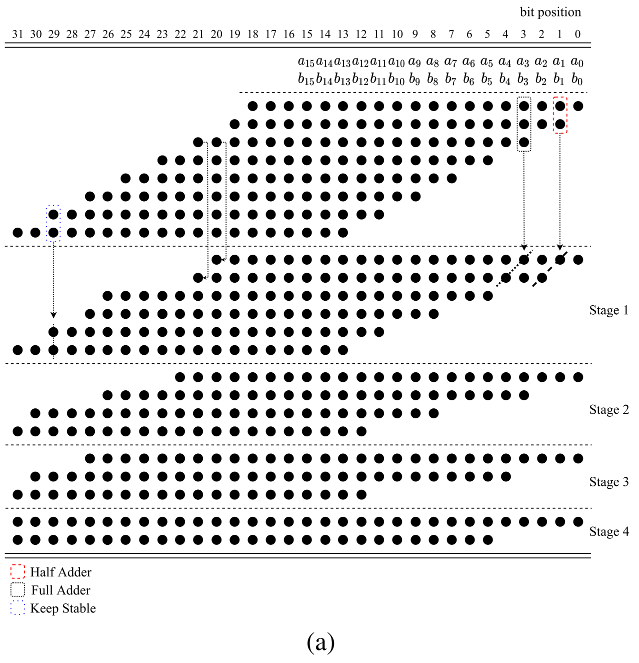
																a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
																b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
													α_2	α_1	α_0	$p_{0,15}$	$p_{0,14}$	$p_{0,13}$	$p_{0,12}$	$p_{0,11}$	$p_{0,10}$	$p_{0,9}$	$p_{0,8}$	$p_{0,7}$	$p_{0,6}$	$p_{0,5}$	$p_{0,4}$	$p_{0,3}$	$p_{0,2}$	$p_{0,1}$	t_0
												1	$\overline{s_1}$	$p_{1,15}$	$p_{1,14}$	$p_{1,13}$	$p_{1,12}$	$p_{1,11}$	$p_{1,10}$	$p_{1,9}$	$p_{1,8}$	$p_{1,7}$	$p_{1,6}$	$p_{1,5}$	$p_{1,4}$	$p_{1,3}$	$p_{1,2}$	$p_{1,1}$	t_1	c_0	
										1	$\overline{s_2}$	$p_{2,15}$	$p_{2,14}$	$p_{2,13}$	$p_{2,12}$	$p_{2,11}$	$p_{2,10}$	$p_{2,9}$	$p_{2,8}$	$p_{2,7}$	$p_{2,6}$	$p_{2,5}$	$p_{2,4}$	$p_{2,3}$	$p_{2,2}$	$p_{2,1}$	t_2	c_1			
								1	$\overline{s_3}$	$p_{3,15}$	$p_{3,14}$	$p_{3,13}$	$p_{3,12}$	$p_{3,11}$	$p_{3,10}$	$p_{3,9}$	$p_{3,8}$	$p_{3,7}$	$p_{3,6}$	$p_{3,5}$	$p_{3,4}$	$p_{3,3}$	$p_{3,2}$	$p_{3,1}$	t_3	c_2					
							1	$\overline{s_4}$	$p_{4,15}$	$p_{4,14}$	$p_{4,13}$	$p_{4,12}$	$p_{4,11}$	$p_{4,10}$	$p_{4,9}$	$p_{4,8}$	$p_{4,7}$	$p_{4,6}$	$p_{4,5}$	$p_{4,4}$	$p_{4,3}$	$p_{4,2}$	$p_{4,1}$	t_4	c_3							
				1	$\overline{s_5}$	$p_{5,15}$	$p_{5,14}$	$p_{5,13}$	$p_{5,12}$	$p_{5,11}$	$p_{5,10}$	$p_{5,9}$	$p_{5,8}$	$p_{5,7}$	$p_{5,6}$	$p_{5,5}$	$p_{5,4}$	$p_{5,3}$	$p_{5,2}$	$p_{5,1}$	t_5	c_4									
		1	$\overline{s_6}$	$p_{6,15}$	$p_{6,14}$	$p_{6,13}$	$p_{6,12}$	$p_{6,11}$	$p_{6,10}$	$p_{6,9}$	$p_{6,8}$	$p_{6,7}$	$p_{6,6}$	$p_{6,5}$	$p_{6,4}$	$p_{6,3}$	$p_{6,2}$	$p_{6,1}$	t_6	c_5											
1	$\overline{s_7}$	$p_{7,15}$	$p_{7,14}$	$p_{7,13}$	$p_{7,12}$	$p_{7,11}$	$p_{7,10}$	$p_{7,9}$	$p_{7,8}$	$p_{7,7}$	$p_{7,6}$	$p_{7,5}$	$p_{7,4}$	$p_{7,3}$	$p_{7,2}$	τ	t_7	c_6													

FIGURE 4. Eight partial products generated by a 16-bit MBE.

Algorithm 2 Pseudo Code for Partial Product Generation of 16-Bit MBE

```
input : Input $\{a_i\}_{i=0}^{15}$, $\{b_i\}_{i=0}^{15}$
output : Partial results $\{p_{i,j}\}_{i=0,j=1}^{7,16}$, $\{s_i\}_{i=0}^{7}$, $\{c_i\}_{i=0}^{6}$, $\tau$, $\alpha_0$,
         $\alpha_1$, $\alpha_2$
variable: $\{na_{i,j}\}_{i=0,j=0}^{7,16}$, $\{neg_i\}_{i=0}^{7}$, $\{one_i\}_{i=0}^{7}$, $\{two_i\}_{i=0}^{7}$,
         $\{t_i\}_{i=0}^{7}$, $\epsilon$, $d$
$neg_0 = b_1$
$\overline{one_0} = \overline{b_0}$
$\overline{two_0} = \overline{b_1 \cdot \overline{b_0}}$
for $i \leftarrow 1$ to 7 do
    $neg_i = (\overline{b_{2i}} + \overline{b_{2i-1}}) \cdot b_{2i+1}$
    $\overline{one_i} = \overline{b_{2i} \oplus b_{2i-1}}$
    $\overline{two_i} = \overline{(\overline{b_{2i+1}} \cdot b_{2i} \cdot b_{2i-1}) + (b_{2i+1} \cdot \overline{b_{2i}} \cdot \overline{b_{2i-1}})}$
for $i \leftarrow 0$ to 6 do
    $c_i = neg_i \cdot (\overline{one_i} + \overline{a_0})$
for $i \leftarrow 0$ to 7 do
    for $j \leftarrow 0$ to 15 do
        $na_{i,j} = \overline{a_j \oplus b_{2i+1}}$
    $na_{i,16} = na_{i,15}$
    for $j \leftarrow 1$ to 16 do
        $p_{i,j} = \overline{(\overline{one_i} + na_{i,j}) \cdot (\overline{two_i} + na_{i,j-1})}$
    $s_i = p_{i,16}$
    $t_i = \overline{\overline{one_i} + \overline{a_0}}$
$\overline{\epsilon} = a_1$ if $\overline{a_0 \cdot b_{15}} = 0$ else $\overline{a_1}$

$\tau = \overline{(\overline{one_7} + \overline{\epsilon}) \cdot (\overline{two_7} + \overline{a_0})}$
$d = \overline{(\overline{b_{15}} + a_0)} \cdot \overline{(b_{13} + a_1) \cdot (b_{14} + a_1) \cdot (b_{14} + b_{13})}$
$\alpha_2 = \overline{s_0 \cdot \overline{d}}$
$\alpha_1 = \overline{\alpha_2}$
$\alpha_0 = \overline{s_0 \oplus \overline{d}}$
```


bit position
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
$a_{15}a_{14}a_{13}a_{12}a_{11}a_{10}\,a_9\,a_8\,a_7\,a_6\,a_5\,a_4\,a_3\,a_2\,a_1\,a_0$
$b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}\,b_9\,b_8\,b_7\,b_6\,b_5\,b_4\,b_3\,b_2\,b_1\,b_0$
Stage 1
Stage 2
Stage 3
Stage 4
Half Adder
Full Adder
Keep Stable
(a)

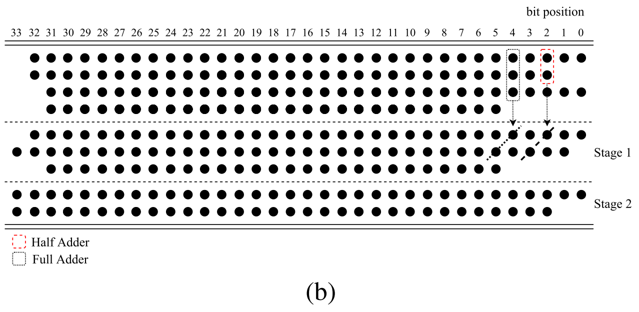


FIGURE 5. (a) Dot diagram of Wallace reduction tree. (b) Dot diagram of 4-to-2 compressor (CP1).

MAC operation. Finally, the proposed BLMAC computes three consecutive multiplications followed by accumulation to generate two partial results, PP_1 and PP_2, and pass them to the next module to get the convolution sum with the 3-D kernel. Note that the product of two 16-bit numbers must be set to 32 bits, and the sum of these 3 products requires an additional 2 bits to avoid quantization errors. Thus, PP_1 and PP_2 are each set to 34 bits. Also, note that Registers 1 and 2 in Fig. 3 use a different clock source than Registers 3 and 4. Specifically, the *word-clk* period is set to be longer than

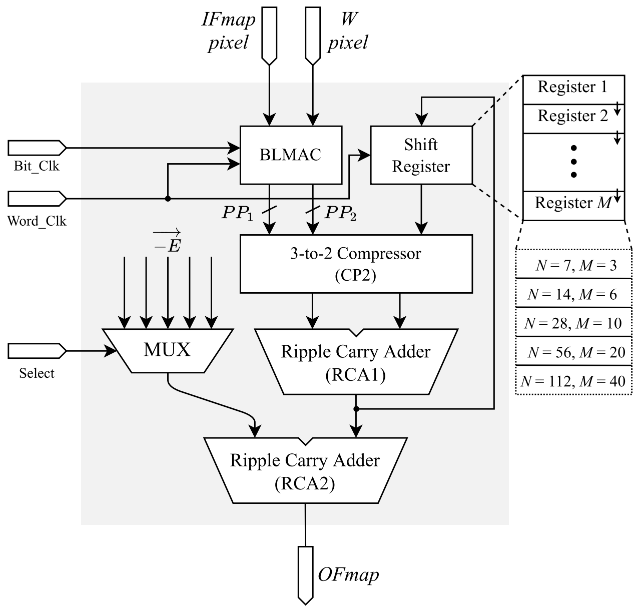


FIGURE 6. **Architecture of the proposed processing element (PE).**

3 times the *bit-clk* period, thus each time when three products are computed, PP_1 and PP_2 are updated.

C. ARCHITECTURE OF PROPOSED PROCESSING ELEMENT

Based on the proposed BLMAC design, a high-speed PE architecture for the CNN is proposed (shown in Fig. 6). The proposed architecture consists of one BLMAC, two RCAs, a shift register, a 3-to-2 compressor (CP2), and a 5-to-1 multiplexer (MUX).

The BLMAC only produces the sum of the three products. Depending on the size of the kernel, the number of BLMAC outputs that must be accumulated to create one output pixel varies. For example, for a $3 \times 3 \times 64$ kernel, the output of BLMAC should be accumulated 192 times. However, they are not generated in successive clock cycles. Therefore, a proper dataflow is required to be envisaged such that the corresponding accumulated value can be obtained from the shift-register successively. In VGG16, as shown in Table 1, the 32-bit product values need to be added up to 4,608 (=$3 \times 3 \times 512$) times. Therefore, the bit width of the shift register is set to 45 bits. The 3-to-2 compressor (CP2) and RCA1 are used to add the two outputs of the BLMAC and the output of the shift register, whose dot diagrams are shown in Figs. 7(a) and 7(b), respectively.

The output PP_1 and PP_2 of WRT (shown in the last stage of Fig. 5(a)), have bit indices of [31:0] and [31:5], respectively. Since the result of a 16-bit multiplication can be represented in 32 bits without overflow, the carry out can be discarded after adding these two partial results. However, in the proposed structure, the two partial results are not added, but are stored separately in two registers of BLMAC in order to limit the critical path. The pair of partial results (PP_1 and PP_2) of BLMAC are added with the results

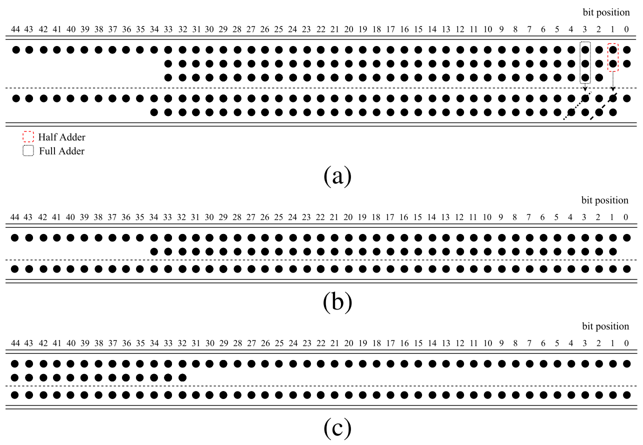


FIGURE 7. **(a) Dot diagram of 3-to-2 compressor (CP2). (b) Dot diagram of RCA1. (c) Dot diagram of RCA2.**

TABLE 2. **Binary representation of the correction vectors used in the five different types of convolution layers.**

-*E*[]	Filter \ Bit	44	43	42	41	40	39	38	37	36	35	34	33	32
-*e*[0]	3×3×3	1	1	1	1	1	1	1	1	0	0	1	0	1
-*e*[1]	3×3×64	1	1	1	0	1	1	1	0	0	0	0	0	0
-*e*[2]	3×3×128	1	1	0	1	1	1	0	0	0	0	0	0	0
-*e*[3]	3×3×256	1	0	1	1	1	0	0	0	0	0	0	0	0
-*e*[4]	3×3×512	0	1	1	1	0	0	0	0	0	0	0	0	0

of the previous results of BLMAC available in the 45-bit shift register. Note that the carry out, which should have been discarded earlier during the addition of PP_1 and PP_2, is accumulated. If the two inputs of MBE have the same sign, then 2^{32} corresponding to the carry out needs to be subtracted. Even if the two signs are different, this subtraction should be done to prevent the sign of the multiplication result from changing due to zero extension. That is, a bias of 2^{32} occurs each time a multiplication is performed, regardless of the two input signs of MBE. The number of multiplications is equal to the kernel size of the convolution layer, and VGG16 has 5 different sized kernels as shown in Table 1. Thus, depending on the convolutional layer being performed, one of the five pre-calculated correction vectors can be selected by a 5:1 MUX and subtracted from the output of the RCA1. The second ripple carry adder in Fig. 6, namely RCA2, performs this subtraction, and the corresponding dot diagram is shown in Fig. 7(c). Table 2 lists the binary representation of correction vectors used in the five different types of convolution layers. For example, for the $3 \times 3 \times 64$ kernel, 576 multiplications are performed to produce one output pixel, so a correction vector corresponding to -576 needs to be added.

IV. DATAFLOW DESIGN AND LATENCY ANALYSIS

The convolution layer consists of as many neurons as the number of pixels in 2-D *OFMap*, and each neuron performs

TABLE 3. The dataflow scheme for convolutional computation for $N = 14$.

Bit	*IFMap*	Convolution Engine			Bit	*IFMap*	Convolution Engine			Bit	*IFMap*	Convolution Engine		
Clock	(I)	PE1	PE2	PE3	Clock	(I)	PE1	PE2	PE3	Clock	(I)	PE1	PE2	PE3
Clk #1	$I_{0,0}\times$	$W_{0,0}$	-	-	Clk #19	$I_{1,0}\times$	$W_{1,0}$	-	-	Clk #37	$I_{2,0}\times$	$W_{2,0}$	-	-
Clk #2	$I_{0,1}\times$	$W_{0,1}$	$W_{0,0}$	-	Clk #20	$I_{1,1}\times$	$W_{1,1}$	$W_{1,0}$	-	Clk #38	$I_{2,1}\times$	$W_{2,1}$	$W_{2,0}$	-
Clk #3	$I_{0,2}\times$	$W_{0,2}$	$W_{0,1}$	$W_{0,0}$	Clk #21	$I_{1,2}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #39	$I_{2,2}\times$	$W_{2,2} \mid O_{0,0}$	$W_{2,1}$	$W_{2,0}$
Clk #4	$I_{0,3}\times$	$W_{0,0}$	$W_{0,2}$	$W_{0,1}$	Clk #22	$I_{1,3}\times$	$W_{1,0}$	$W_{1,2}$	$W_{1,1}$	Clk #40	$I_{2,3}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,1}$	$W_{2,1}$
Clk #5	$I_{0,4}\times$	$W_{0,1}$	$W_{0,0}$	$W_{0,2}$	Clk #23	$I_{1,4}\times$	$W_{1,1}$	$W_{1,0}$	$W_{1,2}$	Clk #41	$I_{2,4}\times$	$W_{2,1}$	$W_{2,0}$	$W_{2,2} \mid O_{0,2}$
Clk #6	$I_{0,5}\times$	$W_{0,2}$	$W_{0,1}$	$W_{0,0}$	Clk #24	$I_{1,5}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #42	$I_{2,5}\times$	$W_{2,2} \mid O_{0,3}$	$W_{2,1}$	$W_{2,0}$
Clk #7	$I_{0,6}\times$	$W_{0,0}$	$W_{0,2}$	$W_{0,1}$	Clk #25	$I_{1,6}\times$	$W_{1,0}$	$W_{1,2}$	$W_{1,1}$	Clk #43	$I_{2,6}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,4}$	$W_{2,1}$
Clk #8	$I_{0,7}\times$	$W_{0,1}$	$W_{0,0}$	$W_{0,2}$	Clk #26	$I_{1,7}\times$	$W_{1,1}$	$W_{1,0}$	$W_{1,2}$	Clk #44	$I_{2,7}\times$	$W_{2,1}$	$W_{2,0}$	$W_{2,2} \mid O_{0,5}$
Clk #9	$I_{0,8}\times$	$W_{0,2}$	$W_{0,1}$	$W_{0,0}$	Clk #27	$I_{1,8}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #45	$I_{2,8}\times$	$W_{2,2} \mid O_{0,6}$	$W_{2,1}$	$W_{2,0}$
Clk #10	$I_{0,9}\times$	$W_{0,0}$	$W_{0,2}$	$W_{0,1}$	Clk #28	$I_{1,9}\times$	$W_{1,0}$	$W_{1,2}$	$W_{1,1}$	Clk #46	$I_{2,9}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,7}$	$W_{2,1}$
Clk #11	$I_{0,10}\times$	$W_{0,1}$	$W_{0,0}$	$W_{0,2}$	Clk #29	$I_{1,10}\times$	$W_{1,1}$	$W_{1,0}$	$W_{1,2}$	Clk #47	$I_{2,10}\times$	$W_{2,1}$	$W_{2,0}$	$W_{2,2} \mid O_{0,8}$
Clk #12	$I_{0,11}\times$	$W_{0,2}$	$W_{0,1}$	$W_{0,0}$	Clk #30	$I_{1,11}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #48	$I_{2,11}\times$	$W_{2,2} \mid O_{0,9}$	$W_{2,1}$	$W_{2,0}$
Clk #13	$I_{0,12}\times$	$W_{0,0}$	$W_{0,2}$	$W_{0,1}$	Clk #31	$I_{1,12}\times$	$W_{1,0}$	$W_{1,2}$	$W_{1,1}$	Clk #49	$I_{2,12}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,10}$	$W_{2,1}$
Clk #14	$I_{0,13}\times$	$W_{0,1}$	$W_{0,0}$	$W_{0,2}$	Clk #32	$I_{1,13}\times$	$W_{1,1}$	$W_{1,0}$	$W_{1,2}$	Clk #50	$I_{2,13}\times$	$W_{2,1}$	$W_{2,0}$	$W_{2,2} \mid O_{0,11}$
Clk #15	$I_{0,14}\times$	$W_{0,2}$	$W_{0,1}$	-	Clk #33	$I_{1,14}\times$	$W_{1,2}$	$W_{1,1}$	-	Clk #51	$I_{2,14}\times$	$W_{2,2} \mid O_{0,12}$	$W_{2,1}$	-
Clk #16	$I_{0,15}\times$	-	$W_{0,2}$	-	Clk #34	$I_{1,15}\times$	-	$W_{1,2}$	-	Clk #52	$I_{2,15}\times$	-	$W_{2,2} \mid O_{0,13}$	-
Clk #17	-	-	-	-	Clk #35	-	-	-	-	Clk #53	-	-	-	-
Clk #18	-	-	-	-	Clk #36	-	-	-	-	Clk #54	-	-	-	-

a MAC operation. In other words, considering the 2-D convolution of the VGG16 convolution layer, a total of $W_O \times H_O$ neurons (number of neurons to generate all the pixels in *OFMap*) is required. Assuming that the neurons read the input pixels in *IFMap* sequentially, one per clock cycle, and performs one multiplication-and-accumulation in a given clock cycle, the number of clock cycles required to get the whole 2-D *OFMap* is $W_I \times H_I \times C_I$ where W_I, H_I, and C_I are the width, height, and number of channels of *IFMap*, respectively. Also, the number of clock cycles required to generate every C_O output channel is $W_I \times H_I \times C_I \times C_O$. When all these neurons are used in parallel, they occupy a large silicon area. In addition, the number of cycles in which one neuron actively performs MAC operations in the entire cycle is $W_W \times H_W \times C_I \times C_O$, which is the size of the 3-D filter multiplied by the number of output channels. Therefore, it can be seen that the utilization of one neuron is significantly low despite the use of a large silicon area.

The reference architecture continuously recycles a limited number of neurons to reduce the silicon area [11]. That is, every pixel in *OFMap* is produced by serially reusing a subset of N neurons. In this case, the number of clock cycles required would be $3 \times (N+2) \times C_I \times C_O \times W_O \times H_O)/N$ for the 3×3 kernel filter. For example, if a subset of $N = 7$ is used in the first convolutional layer, then $3 \times 9 \times 3 \times 64 \times 224 \times 224.7 = 37{,}158{,}912$ clock cycles are consumed while $226 \times 226 \times 3 \times 64 = 9{,}806{,}592$ clock cycles are used in a parallel architecture. This means that the serial architecture increases the latency by 3.79 times compared to the parallel architecture, but can significantly reduce the silicon area. The value of N can be 7, 14, 28, 56, 112, 224 to fit the number of neurons with *OFMap* in the VGG16 architecture. Note that the selection of the size of subset N is a trade-off between silicon area and throughput.

As shown in Fig. 2, the reference architecture consists of three PEs, and every pixel in *OFMap* is created with these three PEs regardless of the value of N. The proposed BLMAC unit speeds up the MAC operation by using the *bit-clk* with a short clock period, while the result of three consecutive MAC operations are synchronized to the *word-clk* with a period of 3 times the *bit-clk* period and stored in the Registers 3 and 4 in Fig. 3. The outputs of Registers 3 and 4 are added with the result of the previous BLMAC operation stored in the shift register in Fig. 6. Considering a hierarchical structure that uses these two clocks with different speeds, the proposed structure requires a different dataflow than that of [11]. As an example, Table 3 shows the dataflow of 3×3.2-D convolution operation of the 11th, 12th, and 13th convolution layers when $N = 14$, that is, Conv($3 \times 3 \times 512.512$). The size of 2-D *IFMap* becomes 16×16 considering zero padding at the border, and the size of 2-D *OFMap* becomes 14×14. Table 3 contains the details of the dataflow generating 14 output samples from the first row of *OFMap*, i.e., from $O_{0,0}$ to $O_{0,13}$.

The BLMAC in the PE1 performs MAC operations over 3 cycles of *bit-clk* #1, #2, #3, and stores the result in the shift register. As a result, the calculations involving the first row of *IFMap* should be stored in *bit-clk* cycle #3, #6, #9, #12, #15 in the case of PE1. Then, the edge of *word-clk* that operates Registers 3 and 4 should also occur accordingly.

TABLE 4. The dataflow scheme for convolutional computation for *N* = 28.

Bit	*IFMap*	Convolution Engine			Bit	*IFMap*	Convolution Engine			Bit	*IFMap*	Convolution Engine		
Clock	(I)	PE1	PE2	PE3	Clock	(I)	PE1	PE2	PE3	Clock	(I)	PE1	PE2	PE3
Clk #1	$I_{0,0}\times$	$W_{0,0}$	-	-	Clk #37	$I_{1,0}\times$	$W_{1,0}$	-	-	Clk #73	$I_{2,0}\times$	$W_{2,0}$	-	-
Clk #2	$I_{0,1}\times$	$W_{0,1}$	$W_{0,0}$	-	Clk #38	$I_{1,1}\times$	$W_{1,1}$	$W_{1,0}$	-	Clk #74	$I_{2,1}\times$	$W_{2,1}$	$W_{2,0}$	-
Clk #3	$I_{0,2}\times$	$W_{0,2}$	$W_{0,1}$	$W_{0,0}$	Clk #39	$I_{1,2}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #75	$I_{2,2}\times$	$W_{2,2} \mid O_{0,0}$	$W_{2,1}$	$W_{2,0}$
Clk #4	$I_{0,3}\times$	$W_{0,0}$	$W_{0,2}$	$W_{0,1}$	Clk #40	$I_{1,3}\times$	$W_{1,0}$	$W_{1,2}$	$W_{1,1}$	Clk #76	$I_{2,3}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,1}$	$W_{2,1}$
Clk #5	$I_{0,4}\times$	$W_{0,1}$	$W_{0,0}$	$W_{0,2}$	Clk #41	$I_{1,4}\times$	$W_{1,1}$	$W_{1,0}$	$W_{1,2}$	Clk #77	$I_{2,4}\times$	$W_{2,1}$	$W_{2,0}$	$W_{2,2} \mid O_{0,2}$
Clk #6	$I_{0,5}\times$	$W_{0,2}$	$W_{0,1}$	$W_{0,0}$	Clk #42	$I_{1,5}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #78	$I_{2,5}\times$	$W_{2,2} \mid O_{0,3}$	$W_{2,1}$	$W_{2,0}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	Clk #79	$I_{2,6}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,4}$	$W_{2,1}$
Clk #15	$I_{0,14}\times$	$W_{0,2}$	$W_{0,1}$	-	Clk #55	$I_{2,0}\times$	$W_{1,0}$	-	-	Clk #80	$I_{2,7}\times$	$W_{2,1}$	$W_{2,0}$	$W_{2,2} \mid O_{0,5}$
Clk #16	$I_{0,15}\times$	-	$W_{0,2}$	-	Clk #56	$I_{2,1}\times$	$W_{1,1}$	$W_{1,0}$	-	Clk #81	$I_{2,8}\times$	$W_{2,2} \mid O_{0,6}$	$W_{2,1}$	$W_{2,0}$
Clk #17	-	-	-	-	Clk #57	$I_{2,2}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #82	$I_{2,9}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,7}$	$W_{2,1}$
Clk #18	-	-	-	-	Clk #58	$I_{2,3}\times$	$W_{1,0}$	$W_{1,2}$	$W_{1,1}$	Clk #83	$I_{2,10}\times$	$W_{2,1}$	$W_{2,0}$	$W_{2,2} \mid O_{0,8}$
Clk #19	$I_{1,0}\times$	$W_{0,0}$	-	-	Clk #59	$I_{2,4}\times$	$W_{1,1}$	$W_{1,0}$	$W_{1,2}$	Clk #84	$I_{2,11}\times$	$W_{2,2} \mid O_{0,9}$	$W_{2,1}$	$W_{2,0}$
Clk #20	$I_{1,1}\times$	$W_{0,1}$	$W_{0,0}$	-	Clk #60	$I_{2,5}\times$	$W_{1,2}$	$W_{1,1}$	$W_{1,0}$	Clk #85	$I_{2,12}\times$	$W_{2,0}$	$W_{2,2} \mid O_{0,10}$	$W_{2,1}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

On the other hand, BLMAC in PE2 produces results after 4th, 7th, 10th, 13th, 16th *bit-clk* cycles, and the PE3 produces results after 5th, 8th, 11th, 14th *bit-clk* cycles. Therefore, the three *word-clk*s that operate PE1, PE2, and PE3 have a phase difference as much as one *bit-clk* period over that of the preceding PE.

If PE1 starts the processing of the second row of *IFMap* (i.e., $I_{1,0}$ to $I_{1,15}$) in *bit-clk* cycle #17, then the first BLMAC result of the second row is generated after *bit-clk* #19. Since the *word-clk* for PE1 is synchronized with the *bit-clk* at Clk #3n in Table 3, the results generated after *bit-clk* #19 cannot be taken from Registers 3 and 4. Therefore, the operation of the second row should start from the 19th *bit-clk*, and the BLMAC result should be scheduled to appear in the 21st *bit-clk*. This output value is added to the one stored in *bit-clk* cycle #3 by CP2 and RCA1 (shown in Fig. 6), and stored back in the shift register. After *bit-clk* #39, a total of 9 MAC operations of 3 × 3 are completed, resulting in the first *OFMap* pixel, $O_{0,0}$. This operation continues and the last pixel in the first row $O_{0,13}$ is generated during the 52nd *bit-clk* cycle.

The proposed dataflow for the subset of $N = 28$ is shown in Table 4 considering that the number of *IFMap* pixels in a row is less than N. As a result, the convolution process of the first row of *IFMap* with the first row of the kernel gets completed using the first 14 neurons. Since 28 neurons are available, to avoid wasting of the rest of the neurons, we use them for the convolution of the second row of *IFMap* (i.e. $I_{1,0}$ to $I_{1,15}$) with the first row of the kernel. Then, the output pixels of the first row of *OFMap* (i.e. $O_{0,0}$ to $O_{0,13}$) are generated between the *bit-clk* cycles #75 and #88.

The length of the shift-register in each PE depends on the value of N. The value of M in Fig. 6 shows the length of the shift register according to each N value. As can be seen from the figure, as N increases, the length of the shift register increases, so the silicon area also increases. On the other hand, as the value of N increases, the number of cycles to complete the convolution process decreases. Appropriate N value can be selected to have the desired trade-off between cost and speed depending on the design objective.

V. DESIGN INNOVATIONS AND CHALLENGES IN THE PROPOSED ARCHITECTURE

A. CRITICAL PATH ANALYSIS

The proposed structure follows a hierarchical design using two clock sources, the *bit-clk* and *word-clk*, and based on precise critical path analysis, we set the timing constraint to $T_{word\text{-}clk} = 3T_{bit\text{-}clk}$. Therefore, precise analysis of the critical paths is required to set appropriate timing constraints for each of the clocks. In this paper, we aim at a high throughput implementation of CNN, and to achieve that it is essential to reduce the $T_{bit\text{-}clk}$ by minimizing the critical path delay of BLMAC. However, if the critical path of the circuit operated by *word-clk* (the circuit outside the BLMAC in Fig. 6) becomes long, the optimization of the BLMAC in terms of throughput becomes ineffective. Therefore, to achieve high throughput we minimize the $T_{bit\text{-}clk}$, and the $T_{word\text{-}clk}$ is designed to be not more than $3T_{bit\text{-}clk}$. A critical path delay model is accordingly evolved to support proposed strategy for high-throughput implementation of CNN.

Modeling the critical path and accurately predicting propagation delays are not straightforward. In particular, since the propagation delay of each gate varies with the temperature and the load capacitance, the actual circuit delay is inevitably different from the delay obtained through the analysis. It is also difficult to predict exact values of delay of the synthesized design because synthesis tool sometimes does minor changes in the selection of library cells during

TABLE 5. Propagation delays on different paths of a half adder and a full adder.

Adder	Path	Notation	Delay	nT_G
Half Adder	A to S	T_{HAS}	T_{XOR}	$2T_{\text{G}}$
	A to C_{OUT}	T_{HAC}	T_{AND}	T_{G}
Full Adder	A to S	T_{FAS}	$2T_{\text{XOR}}$	$4T_{\text{G}}$
	A to C_{OUT}	T_{FAC}	$T_{\text{XOR}} + T_{\text{AND}} + T_{\text{OR}}$	$4T_{\text{G}}$
	C_{IN} to S	T_{FCS}	T_{XOR}	$2T_{\text{G}}$
	C_{IN} to C_{OUT}	T_{FCC}	$T_{\text{AND}} + T_{\text{OR}}$	$2T_{\text{G}}$

various optimization stages. In order to overcome the problem associated with evaluation of the exact propagation delay value, in this paper, propagation delay is not expressed as absolute value in *ns*, but as a relative value as multiple of gate delay (nT_G). During the analysis, we check if the critical path obtained through synthesis report matches our prediction, if the critical path is effectively reduced, and if the timing constraint of $T_{word\text{-}clk} = 3T_{bit\text{-}clk}$ is not violated.

Let us first analyze the critical path in BLMAC operating with *bit-clk* (shown in Fig. 3). The critical path for MBE is the path that computes α_0 (the 17th bit of the first partial product in Fig. 4). When the process of obtaining α_0 from Algorithm 2 is traced back, it can be seen that the delay is the sum of the delays of 2 XOR gates, 1 AND gate, and 1 OR gate. To simplify subsequent analysis, let us define the unit gate delay as T_{G}, and assume that the delays for XOR, AND, and OR gates are $2T_{\text{G}}$, T_{G}, and T_{G}, respectively. Then, the total propagation delay of MBE becomes $6T_{\text{G}}$. Note that this critical path delay is fixed regardless of the input bit-width of the MBE.

Since both WRT and CP1 in Fig. 3 are implemented only with FA and HA, it is also necessary to estimate their delays using T_{G}. The propagation delay from each input to output of FA and HA are summarized in Table 5. A denotes one of the pair of inputs and the output sum and carry-out bits are denoted by S and C_{OUT}, respectively of FA and HA. C_{IN} denotes the carry input of FA. Note that the delay of XOR gate T_{XOR} is taken as $2T_{\text{G}}$ and the delay of AND gate is taken as T_{G}.

The operation of FA in stage of the WRT is not affected by the results of other FAs or HAs in the same stage. Since the FAs and HAs belonging to the same stage can operate simultaneously, the propagation delay of one stage of WRT cannot be greater than the propagation delay of FA. The delay of n-stage WRT therefore can be expressed as

$$T_{\text{WRT}} = nT_{\text{FAS}} = 2nT_{\text{XOR}} = 4nT_{\text{G}}, \tag{2}$$

where n is the number stages of WRT. The delay of 4-stage WRT shown in Fig. 5(a) can be estimated as $16T_{\text{G}}$. Similarly, since the number of stages in CP1 shown in Fig. 5(b) is 2, its delay becomes $8T_{\text{G}}$. The propagation delays of different components of BLMAC and the total propagation delay of BLMAC are listed in Table 6. If the propagation delay of the register is ignored, it takes a total of $30T_{\text{G}}$ to perform one MAC operation. Therefore, if the period of *bit-clk* is set

TABLE 6. Propagation delay of different components of the BLMAC.

Unit	Delay	nT_G
MBE	$2T_{\text{XOR}} + T_{\text{OR}} + T_{\text{AND}}$	$6T_{\text{G}}$
WRT	$4T_{\text{FAS}} = 8T_{\text{XOR}}$	$16T_{\text{G}}$
CP1	$2T_{\text{FAS}} = 4T_{\text{XOR}}$	$8T_{\text{G}}$
Total	$14T_{\text{XOR}} + T_{\text{AND}} + T_{\text{OR}}$	$30T_{\text{G}}$

TABLE 7. Estimate of the propagation delay of the second stage of the PE.

Unit	Delay	nT_G
CP2	$T_{\text{FAS}} = 2T_{\text{XOR}}$	$4T_{\text{G}}$
RCA1	$T_{\text{FAC}} + 32T_{\text{FCC}} + 9T_{\text{HAC}} + T_{\text{HAS}} =$ $2T_{\text{XOR}} + 42T_{\text{AND}} + 33T_{\text{OR}}$	$79T_{\text{G}}$
RCA2	$T_{\text{FAS}} = 2T_{\text{XOR}}$	$4T_{\text{G}}$
Total	$6T_{\text{XOR}} + 42T_{\text{AND}} + 33T_{\text{OR}}$	$87T_{\text{G}}$

to be greater than $30T_{\text{G}}$, the timing constraint would not be violated.

To analyze the critical path of the proposed structure operated by the *word-clk*, it can be seen that the single stage CP2 has 1 FA delay, that is $4T_{\text{G}}$ as shown in the dot diagram of Fig. 7(a). In addition, the delay of RCA1 can be calculated as

$$T_{\text{RCA1}} = T_{\text{FAC}} + 32T_{\text{FCC}} + 9T_{\text{HAC}} + T_{\text{HAS}}, \tag{3}$$

where T_{FAC} is the delay from A to C_{OUT} of FA in bit position 2 in Fig. 7(b). The second term $32T_{\text{FCC}}$ is the delay from C_{IN} to C_{OUT} of FAs in the bit positions 3 to 34. Also, $9T_{\text{HAC}}$ and T_{HAS} are the delay from C_{IN} to C_{OUT} of HAs in the bit positions 35 to 43, and the delay from C_{IN} to S of HA in the bit position 44, respectively. Note that the HA in bit position 1 is not on the critical path because the T_{FCC} in bit position 2 is greater than the sum of T_{HAC} in bit position 1 and T_{FCC} in bit position 2. Using the propagation delays of FA and HA in Table 5, the delay of RCA1 in (3) can be expressed as $79T_{\text{G}}$. Meanwhile, of the two inputs of RCA2, the correction vector can be computed in advance. Once the output S of RCA1 is available at any bit position, the full adder in the same bit position can start the operation. That is, if the output S in the most significant bit position of RCA1 is available, the operation of RCA2 can be completed after one full adder delay of T_{FAS}. Therefore, the propagation delay of RCA2 would be $4T_{\text{G}}$. The sum of the delays for CP2, RCA1, and RCA2, which is equal to $87T_{\text{G}}$ as shown in Table 7, amounts to the critical path delay of the second stage of the PE governed by the *word-clk*.

As mentioned earlier, one *word-clk* period is set to 3*bit-clk* periods. If the *bit-clk* period is set to $30T_{\text{G}}$ as in Table 6, the *word-clk* period can be set to $90T_{\text{G}}$ which is 3 times *bit-clk* period. Note that this setting does not violate the timing constraint since the delay of the critical path using *word-clk* is only $87T_{\text{G}}$, which is shorter than the *word-clk* period.

B. DESIGN INNOVATIONS FOR HIGH-SPEED IMPLEMENTATION

A sequential multiplier requires several clock cycles to perform one multiplication. On the other hand, the Booth algorithm and WRT-based parallel multiplier used in this paper can generate multiplication results every clock cycle, which is essential to achieve high throughput rate. We have used the modified Booth algorithm to effectively reduce the number of PPs in WRT and avoid sign extension, thereby effectively reducing the area as well as the computation-time.

Although a parallel multiplier can perform a multiplication in each clock period, its overall delay is limited by the vector-merging addition of the sum word and the carry word at the last stage of WRT. Since CNN accumulates the results of multiplication multiple times during a convolution operation, it involves a MAC operation with feedback path, making it difficult to utilize pipelining to speed-up the processing. The implementation of pipelining in the feedback path for the MAC operation in the proposed architecture is a critical design challenge but opens up potential for dramatic improvements in speed which is discussed in the following.

- The logic for MAC operation of VGG16 is designed in a hierarchical structure. Specifically, BLMAC in Fig. 6 is driven by *bit-clk* and the other components are subsequently operated by *word-clk*. Through this hierarchical design, the effect of realizing a two-stage pipelined structure in which pipeline registers are inserted after each MAC operation could be obtained. Since VGG16 is based on a 3×3 convolution filter, the proposed architecture is designed in such a way that three consecutive MAC operations become the first pipeline stage, and their accumulation is performed in the second pipeline stage.
- In parallel multipliers, the critical path is lengthened due to the vector-merging addition of the last two reduced words (corresponding to the sum bits and carry bits at the last reduction stage). To overcome this, the last two reduced words are not added but stored separately in two registers. In the subsequent clock cycle, another pair of words are generated and together with the pair of words stored in the registers in the previous cycle form a set of 4 words. This set of 4 words are reduced by a 4-to-2 compressor to produce two reduced words, and prevent an increase in the number of reduced words. Also, two pipeline registers driven by *word-clk* are additionally placed to switch the clock domain between the two pipeline stages. Although there is an additional cost of area loss due to the use of additional resistors, the speed can be greatly improved by this design strategy.
- Critical path analysis is performed so that the propagation delay of the second pipeline stage is three times that of the first, considering the size of 3×3 convolution filter. For this purpose, the propagation delay of each pipeline stage is expressed as the number of gate delays. This allows us to design two pipeline stages for maximum utilization of hardware by suitable choice of the duration of the clock period and maximization of throughput rate.
- The error caused by not adding the partial result pairs PP_1 and PP_2 in BLMAC, is corrected. Specifically, as shown in Fig. 6, it is designed to correct errors using MUX regardless of the size of the convolution filter. Since RCA2, an adder for error correction, is placed adjacent with RCA1, only $4T_G$ amount of additional delay is incorporated by this adder as shown in Table 7. Therefore the proposed architecture increases the throughput rate without compromising with the accuracy.

VI. IMPLEMENTATIONS AND RESULTS

A. SYNTHESIS RESULTS

The proposed architecture is coded at the register transfer level (RTL) using VHSIC hardware description language (VHDL). It was then synthesized by the Synopsys Design Compiler with TSMC 65-nm CMOS standard cell library [24]. During synthesis, the input delay and output delay are set to 0.01 *ns*. The timing constraint is adjusted to find the minimum *bit-clk* period while the slack was maintained positive, and the *word-clk* period is then set to three times the minimum *bit-clk* period. The power consumption is estimated at the clock period which is set to the reciprocal of the maximum usable frequency. We have compared the hardware complexity by implementing circuits where the values of the N were 7, 14, 28, 56, and 112, respectively. In order to demonstrate the effectiveness of the proposed architecture, the state-of-the-art implementation [11] is used for comparison. According to [16], the word lengths of input and kernel weights are quantized to 16-bit, reaching the efficiency of precision and the area. The partial results of the MAC are stored in a 48-bit shift register to prevent overflow even with the maximum convolution size.

The performance of the proposed architecture as obtained from the synthesis results and those of the reference architectures in terms of data arrival time (DAT), maximum usable frequency (MUF), latency, area, area-delay-product (ADP), power consumption, and power-delay-product (PDP) are listed in Table 8. The values of pADP and pPDP are also shown in the table to show the percentage improvement of ADP and PDP compared to the reference architecture, respectively.

It is interesting to find from the synthesis results that the DAT of the proposed architecture is 1.04 *ns*, regardless of the value of N. The DAT is the critical path delay of BLMAC working with the *bit-clk* shown in Table 6. To take full advantage of this short DAT, the *bit-clk* period can be set to the same value as the DAT, thus the MUF up to 961 MHz can be used. The DAT of the proposed structure is only 41.6% compared to the one of the reference architecture, demonstrating the acceleration performance of the proposed structure. In the proposed structure, as the value of N increases, the size of the shift register increases, resulting in an increase in the silicon area. Also, the area of the proposed

TABLE 8. Performance comparison of the proposed architecture and the reference architecture of [11] for different *N* value.

Design	DAT (ns)	MUF (MHz)	Latency (ns)	Area (mm^2)	ADP $(mm^2 \times ns)$	pADP (%)	Power (mW)	PDP $(mW \times ns)$	pPDP (%)
[11] at $N = 7$	2.50	400	16442.9	0.026	0.065	68.5	4.80	12.00	72.7
This work at $N = 7$	1.04	961	6836.6	0.020	0.021		3.16	3.28	
[11] at $N = 14$	2.50	400	14615.8	0.031	0.078	68.9	4.90	12.25	65.7
This work at $N = 14$	1.04	961	6836.6	0.023	0.024		4.04	4.20	
[11] at $N = 28$	2.50	400	13785.0	0.043	0.108	69.2	7.10	17.75	70.6
This work at $N = 28$	1.04	961	5697.4	0.032	0.033		5.02	5.22	
[11] at $N = 56$	2.50	400	13507.2	0.045	0.113	56.7	8.40	21.00	63.1
This work at $N = 56$	1.04	961	5507.6	0.047	0.049		7.45	7.75	
[11] at $N = 112$	2.50	400	13437.0	0.067	0.168	52.2	11.20	28.00	54.2
This work at $N = 112$	1.04	961	5412.9	0.077	0.080		12.33	12.82	

structure is reduced compared to the reference when N is 7, 14, and 28, but larger when N is 56 and 112. However, since the DAT is greatly reduced, the proposed architecture has an advantage in terms of ADP for any of the values of N. Specifically, ADP can be reduced by 52.2% to 69.2% depending on the value of N.

The power consumption of the proposed architecture is also less than that of the reference architecture, except for $N = 112$. Although power consumption varies depending on the clock period, the power-delay-product (PDP) obtained by multiplying power consumption and the delay tends to remain constant. Therefore, in this paper, PDP is used as a comparison metric with existing structures. Specifically, the proposed architecture has at least 54.2% less PDP than the reference architecture. Therefore, it can be seen that the proposed structure not only provides significantly better efficiency in terms of area-delay product but also the power consumption.

Power consumption is affected not only by the frequency of the operating clock, but also by the toggle rate (*TR*) and the probability that the input is in logic state 1 (P_1 probability). Most papers provide the power estimated at operating clock frequency, but do not provide the *TR* or P_1 values, thus it is difficult to make a fair comparison with other papers in terms of power consumption. In this paper, we set $P_1 = 0.1$ and $TR = 0.1$ times clock frequency, which are reasonable because they are the default values set by general synthesis tools. In the proposed architecture, the power consumption can be effectively reduced by gating of the RCA2 input used for the correction at the end of the overall calculation of the 3D kernel.

According to the dataflow of the proposed architectures, the delay of each convolution set of the reference and proposed architectures can be expressed as

$$d_{ref} = \begin{cases} \dfrac{3 \times (N + \lceil N/W_O \rceil \times 2) \times C_I \times C_O \times W_O \times H_O}{N \times F}, & N > W_O \\ \dfrac{3 \times (N + 2) \times C_I \times C_O \times W_O \times H_O}{N \times F}, & \text{otherwise} \end{cases} \tag{4}$$

$$d_{prop} = \begin{cases} \dfrac{3 \times (N + 4) \times C_I \times C_O \times W_O \times H_O}{N \times F}, & N = 14 \\ \dfrac{3 \times (N + 2) \times C_I \times C_O \times W_O \times H_O}{N \times F}, & \text{otherwise} \end{cases} \tag{5}$$

where C_I, C_O, W_O, H_O, N, and F are the number of channels of *IFMap*, the number of channels of *OFMap*, the width of *OFMap*, the height of *OFMap*, the size of subset, and the clock frequency, respectively. Note that F in the proposed architecture means the maximum usable frequency of *bit-clk*. The computation time of each convolution set with different values of N are listed in Tables 9 and 10. It can be seen that the latency of both structures decreases as the value of N increases. In particular, since the clock period of the proposed structure is significantly shorter than that of the reference architecture, the latency required to complete the operations in entire convolution layers is also significantly reduced.

B. APPLICATION OF THE PROPOSED ARCHITECTURE

The purpose of the proposed design is to achieve significant increase of throughput using gate-level pipelining. When the proposed architecture is intended to be used as a computing core for an existing neural network, it can be flexibly adjusted according to changes in the kernel size, bit width, and N value. The flexible realization of proposed architecture to fit in to the design specifications of CNN as follows:

- For shorter bit-width implementation, pruning can be applied to the Wallace tree in Figs. 5 and 7 of the proposed structure, and conversely, for a longer bit width implementation, bit extension can be applied to the Booth encoder and the Wallace tree.
- For a generic $W \times W$ convolution kernel other than 3×3, in the BLMAC in Fig. 3, the transfers from Registers 1 and 2 to Registers 3 and 4 are to be performed after W times accumulation instead of 3 times. When W is greater than 3 and the cumulative count is incremented, sign-extension is applied in the Wallace tree to prevent overflow. In this case, the timing constraint is not violated if the period of *word-clk* is set to $W \times$ *bit-clk*.

TABLE 9. Computation time of VGG16 convolutional layer of the reference architecture [11] for input image of size 224 × 224.

Reference VGG16	C_I	C_O	W_O	H_O	N=7 (ms)	N=14 (ms)	N=28 (ms)	N=56 (ms)	N=112 (ms)
Conv(3×3×3, 64)	3	64	224	224	92.9	82.6	77.4	74.8	73.5
Conv(3×3×64, 64)	64	64	224	224	1981.8	1761.6	1651.5	1596.5	1568.9
Conv(3×3×64, 128)	64	128	112	112	990.9	880.8	825.8	798.2	784.5
Conv(3×3×128, 128)	128	128	112	112	1981.8	1761.6	1651.5	1596.5	1568.9
Conv(3×3×128, 256)	128	256	56	56	990.9	880.8	825.8	798.2	798.2
2×Conv(3×3×256, 256)	256	256	56	56	3963.6	3523.2	3303.0	3193.0	3193.0
Conv(3×3×256, 512)	256	512	28	28	990.9	880.8	825.8	825.8	825.8
2×Conv(3×3×512, 512)	512	512	28	28	3963.6	3523.2	3303.0	3303.0	3303.0
3×Conv(3×3×512, 512)	512	512	14	14	1486.5	1321.2	1321.2	1321.2	1321.2
Total					16442.9	14615.8	13785.0	13507.2	13437.0

TABLE 10. Computation time of VGG16 convolutional layer of the proposed architecture for the input image of size 224 × 224.

Proposed VGG16	C_I	C_O	W_O	H_O	N=7 (ms)	N=14 (ms)	N=28 (ms)	N=56 (ms)	N=112 (ms)
Conv(3×3×3, 64)	3	64	224	224	38.6	38.6	32.2	31.1	30.6
Conv(3×3×64, 64)	64	64	224	224	824.0	824.0	686.7	663.8	652.4
Conv(3×3×64, 128)	64	128	112	112	412.0	412.0	343.3	331.9	326.2
Conv(3×3×128, 128)	128	128	112	112	824.0	824.0	686.7	663.8	652.4
Conv(3×3×128, 256)	128	256	56	56	412.0	412.0	343.3	331.9	326.2
2×Conv(3×3×256, 256)	256	256	56	56	1648.0	1648.0	1373.4	1327.6	1304.8
Conv(3×3×256, 512)	256	512	28	28	412.0	412.0	343.3	331.9	326.2
2×Conv(3×3×512, 512)	512	512	28	28	1648.0	1648.0	1373.4	1327.6	1304.8
3×Conv(3×3×512, 512)	512	512	14	14	618.0	618.0	515.1	498.0	489.3
Total					6836.6	6836.6	5697.4	5507.6	5412.9

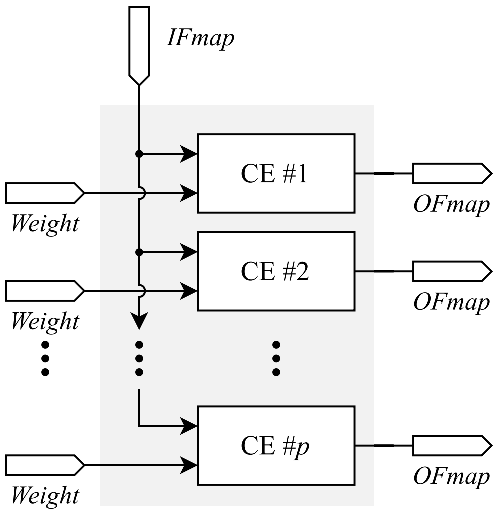


FIGURE 8. The high-level architecture of the proposed parallel convolution engine.

- If the value of N changes, the size of the shift-register in Fig. 6 can be changed accordingly.

Based on the throughput requirement of a specific CNN, the parallelism of the convolution engine can be modified as shown in Fig. 8. Let us denote the parallelism index as p. Each CE produces the *OFmap* of one output channel. Also, each CE has the same *IFmap* but receive different input *Weights* from different input filters. In the parallel architecture, latency of computation is decreased by a factor p, however the area and power consumption are increased by almost the same factor p. The synthesis results of proposed parallel architecture with $p = 32$ and some of the existing CNN accelerator architectures are shown in Table 11.

The architecture of Moons *et al.* [25] has been synthesized using a more recent fully depleted silicon on insulator (FD-SOI) technology library than the others. The work of [25] uses the dynamic fixed-point to perform CNN with flexible bit-width ranging from 1 to 16 bits based on the precision requirement of the application. When running VGG16, this architecture provides a throughput of 1.67 fps and consumes just 26 mW power, but our design involves 13.03% less gate-counts and provides 3.53 times higher throughput. The proposed hardware accelerator has 2.57 times higher throughput using about 1.77 times less PDP than the MA-efficient design of [11]. It is also worth mentioning that the proposed accelerator provides the peak performance at 181 Giga operations per second (GOPS) which is 2.38 times higher than that of [11]. The architecture of Chen *et al.* [18] which is implemented in 65 nm CMOS technology involves 1.67 times less power consumption using data compression and network sparsity techniques, but the proposed CNN hardware accelerator outperforms the work of [18] in terms of throughput (8.4× faster), latency (25.5× lower), gate count (8.4% less) with (3.8× higher) nominal frequency.

TABLE 11. Performance comparison of the reference and the proposed architectures.

VGG16 Architectures	Moons *et al* [25]	Chen *et al* [18]	Ardakani *et al* [11]		This work
Technology	28 nm	65 nm	65 nm		65 nm
Methodology	Silicon	Silicon	Synthesis		Synthesis
Type	–	–	Area-efficient	MA-efficient	–
Core Area (mm^2)	1.87	12.52	1.77	3.5	2.5
Power (mW)	26	236	254	260	393.6
Total Latency (ms)	598.8	4309.5	436.4	453.3	169.1
Throughput (fps)	1.67	0.7	2.3	2.2	5.9
# MAC	256 (16b)-1024 (4b)	168	96	192	96
Nominal Frequency (MHz)	200	250	400	200	962
Peak Performance (Gops)	102 (16b)-408 (4b)	84	76	76	181
Bitwidth (bits)	1-16 (flexible)	16 fixed	16 fixed		16 fixed
Filter-sizes	all	1-12(h), 1-32(v)	3×3		3×3

VII. CONCLUSION

In this paper, we have proposed a novel bit-level MAC design and PE architecture for high-speed hardware implementation of the CNN accelerator. The proposed structure can greatly improve the speed by the proposed bit-level design using modified two stage Wallace reduction and modified Booth recoding by reducing the computational delay of MAC operation. The proposed design works with dual-clock strategy where the MAC operations are accelerated with a faster clock while their accumulation for convolution operation operates with a longer clock period, in a two-stage hierarchical design. Precise critical path analysis is performed to set timing constraints appropriately in the hierarchical structure. The inevitable errors due to the hierarchical addition, the correction vectors were calculated on the sideline concurrently, and added to compensate for the errors. Dataflow and latency analysis of the proposed structure have been demonstrated for the convenience of the readers. The proposed structure has been experimentally proven to be efficient in terms of area, throughput, and power consumption by showing that it outperforms the existing architectures in terms of ADP and PDP. Therefore, the proposed structure is suitable for implementing the CNN for high-speed real-time applications. In addition, the proposed structure can be applied to various CNN algorithms with different network architectures based on MAC computation.

ACKNOWLEDGMENT

The EDA Tool was supported by the IC Design Education Center (IDEC), Korea. A part of this work is done in collaboration with Sandhaan Labs Private Limited, Bhubaneswar, India.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.

[3] P. Napoletano, F. Piccoli, and R. Schettini, "Anomaly detection in nanofibrous materials by CNN-based self-similarity," *Sensors*, vol. 18, no. 2, p. 209, Jan. 2018.

[4] C. Eggert, S. Brehm, A. Winschel, D. Zecha, and R. Lienhart, "A closer look: Small object detection in faster R-CNN," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2017, pp. 421–426.

[5] L. Chen, S. Wang, W. Fan, J. Sun, and S. Naoi, "Beyond human recognition: A CNN-based framework for handwritten character recognition," in *Proc. 3rd Asian Conf. Pattern Recognit. (ACPR)*, Nov. 2015, pp. 695–699.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[7] E. Vansteenkiste, "New FPGA design tools and architectures," Ph.D. dissertation, Dept. Electron. Inf. Syst., Ghent Univ., Ghent, Belgium, 2016.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1–9.

[9] 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.

[10] J. Jo, S. Kim, and I.-C. Park, "Energy-efficient convolution architecture based on rescheduled dataflow," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4196–4207, Dec. 2018.

[11] A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, "An architecture to accelerate convolution in deep neural networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 65, no. 4, pp. 1349–1362, Apr. 2018.

[12] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, Apr. 2015, pp. 1–14.

[14] A. Albert, J. Kaur, and M. C. Gonzalez, "Using convolutional networks and satellite imagery to identify patterns in urban environments at a large scale," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1357–1366.

[15] Q. Liu, C. Feng, Z. Song, J. Louis, and J. Zhou, "Deep learning model comparison for vision-based classification of full/empty-load trucks in earthmoving operations," *Appl. Sci.*, vol. 9, no. 22, p. 4871, Nov. 2019.

[16] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[17] M. Alawad and M. Lin, "Scalable FPGA accelerator for deep convolutional neural networks with stochastic streaming," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 4, pp. 888–899, Oct. 2018.

[18] 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, Nov. 2017.

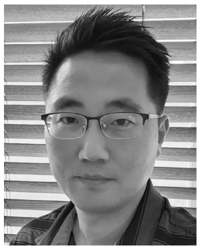
[19] B. Moons and M. Verhelst, "An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, Apr. 2017.

[20] J. Xu, Y. Huan, B. Huang, H. Chu, Y. Jin, L.-R. Zheng, and Z. Zou, "A memory-efficient CNN accelerator using segmented logarithmic quantization and multi-cluster architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 6, pp. 2142–2146, Jun. 2021.
[21] Y.-L. Wu, Y. Lu, and J.-D. Huang, "High-speed power-efficient coarse-grained convolver architecture using depth-first compression scheme," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.
[22] S. R. Kuang, J. P. Wang, and C. Y. Guo, "Modified Booth multipliers with a regular partial product array," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 5, pp. 404–408, May 2009.
[23] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.
[24] *TSMC 65 nm CMOS Logic General Purpose Plus CMOS Standard Cell Libraries—tcbn65gplus*.
[25] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

SUN SIK LEE received the B.S. and M.S. degrees in electronic engineering from Myongji University, Yongin, Republic of Korea, in 2018 and 2021, respectively. Since 2021, he has been working as an Engineer at YIK Corporation, Seongnam, Republic of Korea. His research interests include low power digital system design, high-speed video signal processing, and machine learning.

THANH DAT NGUYEN received the B.S. degree in electronic engineering from the HCMC University of Technology and Education (HCMUTE), Ho Chi Minh, Vietnam, in 2018, and the M.S. degree in electronic engineering from Myongji University, Yongin, Republic of Korea, in 2021. Since 2021, he has been working as an Engineer at OCST Corporation, Seoul, Republic of Korea. His research interests include high-speed digital system design, deep learning applications, and image processing.

PRAMOD KUMAR MEHER (Senior Member, IEEE) received the B.Sc. (Hons.) and M.Sc. degrees in physics and the Ph.D. degree in science from Sambalpur University, Sambalpur, India, in 1976, 1978, and 1996, respectively. He is currently with C. V. Raman Global University, Bhubaneswar, India, as a Senior Professor. He was with the School of Computer Science and Engineering, Nanyang Technological University, Singapore, in senior research positions, from 2005 to 2016. Previously, he was a Reader in electronics with Berhampur University, Berhampur, India, from 1993 to 1997, and a Professor in computer applications with Utkal University, Bhubaneswar, from 1997 to 2002. He has also worked as a Technical Consultant with the Cyber Security Research Centre, Nanyang Technological University, Singapore. He has founded Sandhaan Labs Pvt. Ltd., for promoting research and development on computer engineering and applications and for the industry-academic collaborations. He has contributed nearly 250 technical articles to various reputed journals and conference proceedings, including nearly 90 articles in various IEEE Transactions. He is the Co-Editor of the book *Arithmetic Circuits for DSP Applications* (Wiley-IEEE Press). His current research interests include signal processing, cyber security, intelligent computing for smart systems, the IoT, and analytics. He is a fellow of the Institution of Electronics and Telecommunication Engineers, India. He was a recipient of the Samanta Chandrasekhar Award for excellence in Research on Engineering and Technology, in 1999. He was a Speaker for the Distinguished Lecturer Program of the IEEE Circuits Systems Society, from 2011 to 2012. He has served as an Associate Editor for the IEEE Transactions on Circuits and Systems for Video Technology, the IEEE Transactions on Circuits and Systems—II: Express Briefs, the IEEE Transactions on Circuits and Systems—I: Regular Papers, and the IEEE Transactions on Very Large Scale Integration Systems. He is currently an Associate Editor of the IEEE Transactions on Circuits and Systems—I: Regular Papers and the *Circuits, Systems, and Signal Processing* journal.



SANG YOON PARK (Member, IEEE) received the B.S. degree in electrical engineering and the M.S. and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, Republic of Korea, in 2000, 2002, and 2006, respectively. He joined the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, as a Research Fellow, in 2007. From 2008 to 2014, he was a Research Scientist with the Institute for Infocomm Research, Singapore. Since 2014, he has been with the Department of Electronics Engineering, Myongji University, Yongin, Republic of Korea, where he is currently an Associate Professor. His research interests include design of dedicated and reconfigurable architectures for low-power and high-performance digital communication systems.