Article

Not peer-reviewed version

Efficient Two-Stage Max-Pooling Engines for an FPGA-based Convolutional Neural Network

Eonpyo Hong * , Kang-A Choi , And Jhihoon Joo

Posted Date: 10 August 2023

doi: 10.20944/preprints202308.0825.v1

Keywords: max-pooling; convolutional neural network (CNN); FPGA; rank tracking based max-pooling (RTB-MAXP); cascaded maximum based max-pooling (CMB-MAXP)

Article

Efficient Two-Stage Max-Pooling Engines for an FPGA-based Convolutional Neural Network

Eonpyo Hong *, Kang-A Choi and Jhihoon Joo

Agency for Defense Development; Daejeon 34186, Republic of Korea; kachoi@add.re.kr (K.C.); jhihoon@add.re.kr (J.J.)

* Correspondence: ephong77@gmail.com or ephong77@add.re.kr; Tel.: +82-42-821-2643

Abstract: This paper proposes two max-pooling engines, named the RTB-MAXP engine and the CMB-MAXP engine, with a scalable window size parameter for FPGA-based convolutional neural network (CNN) implementation. The max-pooling operation for the CNN can be decomposed into two stages, i.e., a horizontal axis max-pooling operation and a vertical axis max-pooling operation. These two one-dimensional max-pooling operations are performed by tracking the rank of the values within the window in the RTB-MAXP engine and cascading the maximum operations of the values in CMB-MAXP engine. Both the RBM-MAXP engine and the CMB-MAXP engine were implemented using VHSIC Hardware Description Language (VHDL) and verified by simulations. They have been employed for and tested in our CNN accelerator targeting at the CNN model YOLOv4-CSP-S-Leaky for object detection.

Keywords: max-pooling; convolutional neural network (CNN); FPGA; rank tracking based max-pooling (RTB-MAXP); cascaded maximum based max-pooling (CMB-MAXP)

1. Introduction

Convolutional Neural Networks (CNNs) have demonstrated remarkable performance in various domains, including image classification, object detection, and speech recognition [1,2]. However, effectively integrating CNNs into embedded systems with limited power and size requirements remains a significant challenge. This is primarily due to the high computational demands of CNNs, which can be resource-intensive for embedded systems. Typically, embedded systems revolve around general-purpose central processing units (CPUs) capable of handling a wide range of tasks. However, CPUs have limitations when it comes to implementing CNNs, mainly because of the repetitive and computationally intensive nature of large-scale convolution operations. To address this challenge and efficiently implement CNNs, dedicated hardware accelerators such as graphic processing units (GPUs) or field-programmable gate arrays (FPGAs) are commonly employed [3]. Among these options, FPGAs have gained popularity for implementing CNNs in embedded systems, primarily due to their ability to perform convolution operations in parallel with high-energy efficiency. Compared to GPUs, FPGAs offer higher energy efficiency, making them an attractive choice for resource-constrained embedded systems [4–6].

The generation of feature maps, achieved through the fundamental convolution operation using multiple kernels, plays a crucial role in CNNs. To minimize computational costs and simplify the model, reducing the size of these feature maps is necessary. The max-pooling technique is employed to achieve this while preserving spatial invariance of distinct features within the feature maps [7,8]. Typically, a window of size 2×2 is used in max-pooling operation, ensuring spatial overlap of the maximum values, and sampling values along the horizontal and vertical axes every 2 positions [8]. As a result, the feature map's width and height can be reduced by half, resulting in a 4x reduction in size while preserving the maximum values that represent the distinct features of the feature map. In recent years, there has been a growing focus on improving object detection performance by utilizing feature maps of various resolutions. This approach often involves incorporating max-pooling techniques with larger and diverse window sizes [9]. For instance, YOLOv4's Spatial Pyramid Pooling (SPP) employed the max-pooling with window sizes of 5, 9, and 13 [9–11]. By employing different window sizes, the pooling operation captures multi-scale information from the feature maps, enabling the model to detect objects at different sizes and scales more effectively. This approach improves object detection accuracy and robustness in complex scenes [9–11].

The 2x2 max-pooling operation was simply implemented by using two delay buffers and three comparators on FPGAs [12,13]. This 2×2 max-pooling engine was then extended to accommodate the max-pooling with the *k*×*k* window by utilizing *k* delay buffers and *k*×*k*-1 comparators [14,15]. However, despite its simplicity, the *k*×*k*-1 comparator-based max-pooling engine is inefficient in terms of hardware utilization due to the requirement of numerous comparators with large windows. Additionally, it results in high power consumption due to the usage of many logic cells. In order to reduce the required comparators, we propose two efficient max-pooling engines named the rank tracking based max-pooling (RTB-MAXP) engine and the cascaded maximum based max-pooling (CMB-MAXP) engine. The two-dimensional max-pooling operation is decomposed into the horizontal max-pooling operation and the vertical max-pooling operation for the operational efficiency. Thus, the two-dimensional max-pooling operation can be accomplished with the two-step one-dimensional max-pooling operation, which leads to the reduction of comparison operations from *k*×*k*-1 to 2*k*-2. In the RTB-MAXP engine, the max-pooling operation is accomplished by tracking the ranks of the values within the scalable window and extracting the top ranked value as the maximum value. On the other hand, the CMB-MAXP engine employs the cascaded maximum operations to find the maximum value within the window.

This paper is organized as follows: In section 2, the two-dimensional max-pooling operation is represented into the form of two-stage max-pooling operations, and then the architecture of the RTB-MAXP engine is introduced in Section 3. Section 4 describes the architecture of the proposed CMB-MAXP engine. Section 5 shows the implementation results of the RTB-MAXP engine and CMB-MAXP engine and then those are compared in terms of resource utilization. Finally, the conclusion is provided in Section 6.

2. Max-Pooling Operation

The max-pooling is a fundamental operation in deep learning used for reducing the spatial dimensions of a feature map. It involves sliding a window over the feature map and selecting the maximum value within the window [5]. Since the values in the feature map represent features, it is crucial to avoid losing important large values when reducing the size of the feature map. The max-pooling ensures that the maximum values within the window are duplicated and remained, preventing the loss of maximum values during spatial sampling. Given an input feature map x, containing three-dimensional data of P channels, width W, and height H, the output of the max-pooling operation z can be expressed as the following equation:

$$z_p(i,j) = \max_{\left\lfloor -\frac{k}{2}\right\rfloor \le l,m \le \left\lfloor \frac{k-1}{2}\right\rfloor} x_p(i+l, j+m), \quad (1)$$

where p ($0 \le p \le P$-1) represents the index of the channel, i and j ($0 \le i \le H$-1, $0 \le j \le W$-1) denote the vertical and horizontal indices of the feature map, the operator "max" represents the maximum operation, and the operator $\lfloor \cdot \rfloor$ represents a floor operation, respectively. The value of k determines the size of the k×k square window. Note that the subsampling reducing the spatial size of feature maps z is not considered in this paper because it can easily and simply be implemented.

The two-dimensional max-pooling operation in (1) can be decomposed into two distinct one-dimensional max-pooling operations as follows:

$$y_p(i,j) = \max_{\left\lfloor -\frac{k}{2}\right\rfloor \le m \le \left\lfloor \frac{k-1}{2}\right\rfloor} x_p(i, j+m) \quad (2)$$

$$z_p(i,j) = \max_{\left\lfloor -\frac{k}{2}\right\rfloor \le l \le \left\lfloor \frac{k-1}{2}\right\rfloor} y_p(i+l, j) \quad (3)$$

Operations in (2) and (3) can be named as the horizontal axis max-pooling operation and the vertical axis max-pooling operation, respectively. The output of the horizontal max-pooling operation is fed to the input of the vertical max-pooling operation. Using these two sequential max-pooling operations reduces the number of the comparison operations compared to the two-dimensional max-pooling operation in (1), i.e. from $k\times k$-1 to 2k -2.

3. Rank Tracking Based Max-Pooling (RTB-MAXP) Engine

In this section, a two-step max-pooling operation engine, named RTB-MAXP engine, is introduced. This engine is designed on the basis of the rank tracking concept to find the maximum value within a kernel window as shown in Figure 1. It has registers that store the sample values within a window in descending order. The maximum value is stored in register r_0, and the values within the window are stored in descending order down to r_{K-1}, where K is the maximum window size. When a new value of the feature map is input to the engine, the registers need to be updated. For example, if the new input value $x_p(i,j)$ is smaller than r_0 but larger than r_1, the value in r_3 is the value pushed out of the window, the value of r_2 is shifted to r_3, the value of r_1 is shifted to r_2, and the value $x_p(i,j)$ is stored in r_1 simultaneously. Meanwhile the values of r_0 and r_4,... ,r_{K-1} remain unchanged. The resulting output of this one-dimensional max-pooling engine corresponds to the value stored in r_0. This rank tracking concept is implemented through two distinct blocks: the rank-counting block and the delay-counting block.

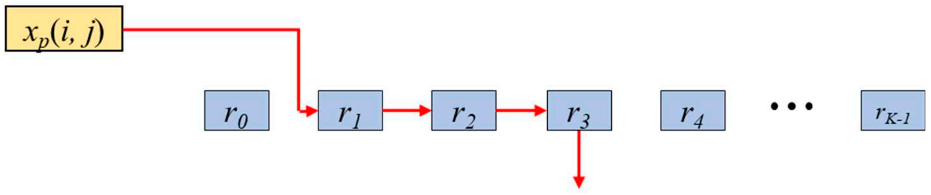


Figure 1. Operating principle of the horizontal max-pooling engine with maximum possible window size K when incoming value $x_p(i,j)$ is $r_0 < x_p(i,j) < r_1$ and the r_3 is pushed out of the window.

The rank-counting block is for storing the input values into a corresponding register based on their ranks. It is comprised of K blocks named R_v (v=0, 1, ..., K-1) and the comparator blocks denoted as '>', as shown in Figure 2. Each R_v block is composed of a multiplexer, a multiplexer switch (MS), and a register r_v as depicted in Figure 3. The inputs for the multiplexer include the input feature map value $x_p(i,j)$, the value of r_v, the value of r_{v-1} (one step larger value than r_v), and the value of r_{v+1} (one step smaller than r_v). The multiplexer selects one out of these inputs based on the MS's output values c_v[0] and c_v[1]. Table 1 shows the relationship between input values of m_{v-1}, m_v, m_{v+1}, and n_v, and the corresponding output values c_v[0] and c_v[1]. The values of m_{v-1}, m_v, and m_{v+1} are obtained from the outputs of the comparators shown in Figure 2, while the value of n_v is obtained from the delay-counting block. The comparator block outputs '1' when $x_p(i, j)$ is larger than r_v, and '0' otherwise.

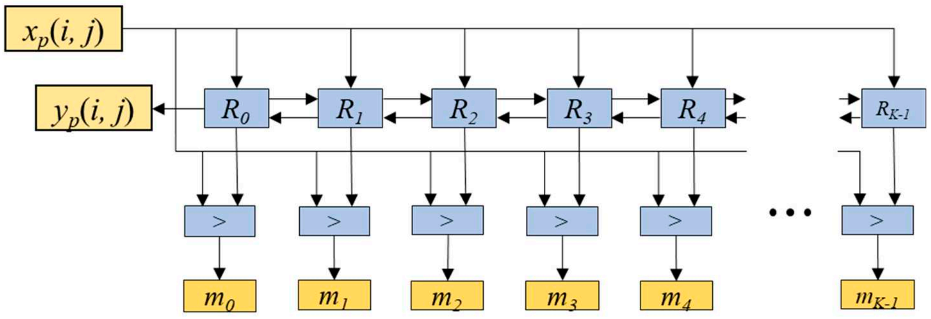


Figure 2. Rank-counting block consisting of R_v blocks (v=0, 1, ..., K-1) and comparator blocks marked as '>'.

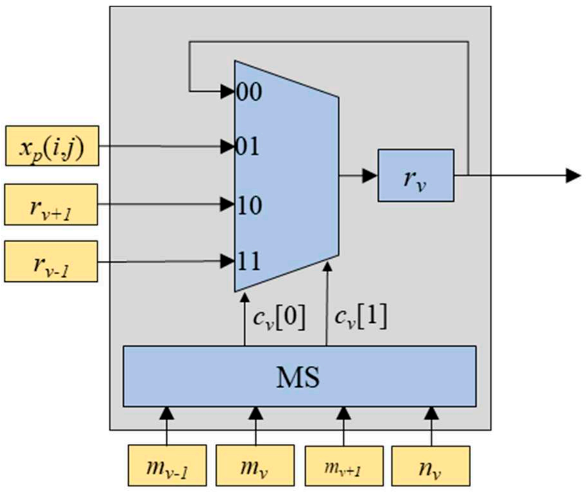


Figure 3. Structure of R_v block consisting of the multiplexer, the multiplexer switch (MS), and the register r_v.

Table 1. The relationships between the multiplexer switch (MS)'s input and output, and the output of the multiplexer exploited as R_v and D_v.

Multiplexer Switch (MS)						Multiplexer	
Input				Output		Output	
m_{v-1}	m_v	m_{v+1}	n_v	$c_v[0]$	$c_v[1]$	r_v	d_v
0	0	X	0	0	0	r_v	d_v+1
0	1	X	0	0	1	$x_p(i,j)$	0
0	1	X	1	0	0	r_v	d_v+1
1	0	X	0	0	0	r_v	d_v+1
1	1	X	0	1	0	r_{v+1}	$d_{v+1}+1$
1	1	X	1	0	0	r_v	d_v+1
X	0	0	1	1	1	r_{v-1}	$d_{v-1}+1$
X	0	1	1	0	1	$x_p(i,j)$	0

The value pushed out of the window is also required to be tracked for the max-pooling operation, for which the delay-counting block is employed. The delay-counting block indicates the value n_v (v=0, 1,…, K-1) which is pushed out of the window. It consists of the K blocks named D_v (v=0, 1, ..., K-1), the comparator blocks denoted as '=', and OR gates, as depicted in Figure 4. The block D_v outputs the delay indicating value corresponding to the ranking-counting value r_v. Similar to the ranking-counting block, the block D_v is composed of a multiplexer, a MS, and a register denoted as 'd_v', as shown in Figure 5. The inputs for the multiplexer include the increased delay-counting value d_v+1 corresponding to r_v, the delay-counting value 0 corresponding to $x_p(i,j)$, the increased delay-counting value d_{v-1}+1 corresponding to r_{v-1}, the increased delay-counting value d_{v+1}+1 corresponding to r_{v+1}. The multiplexer selects one out of these inputs based on the MS's output values $c_v[0]$ and $c_v[1]$. Note that the MS is the same as that in block R_v. As shown in Figure 4, the comparator block outputs 1 if the delay-counting value is equal to the selected window size k, and 0 otherwise. This makes it possible to determine which value is pushed out of the window. The value n_v is the output of the OR operation between n_{v-1} and the output of the comparator block as shown in Figure 4.

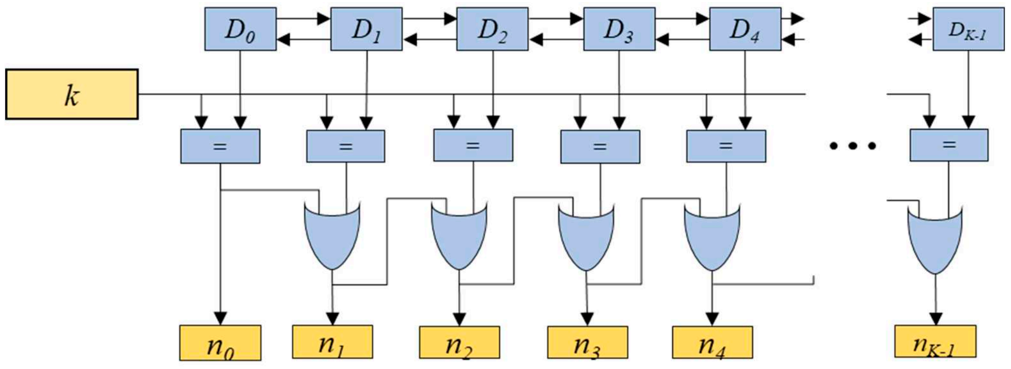


Figure 4. Delay-counting block consisting of D_v (v=0, 1, ..., K-1), comparators marked as '=', and OR gates.

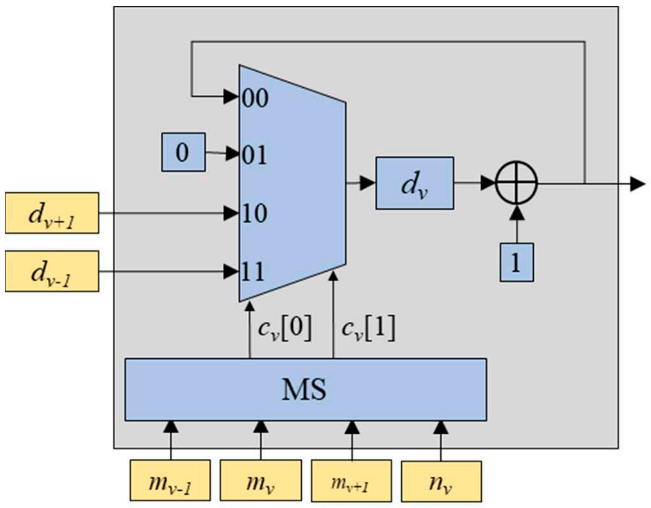


Figure 5. Structure of D_v consisting of multiplexer, multiplexer switch (MS), and the register d_v.

The two-dimensional max-pooling engine is implemented by employing the multiple one-dimensional rank tracking based max-pooling engines as depicted in Figure 6. The block marked with "M^H" represents the horizontal one-dimensional max-pooling engine shown in equation (2). Specifically, $y_p(i,j)$ is obtained from the highest-ranking value r_0 of the ranking-counting block illustrated in Figure 2. In order to obtain $z_p(i,j)$, where $0 \le j \le W$-1, each column of $y_p(i,j)$ is then fed into W vertical max-pooling engines $M^V(j)$. Then, the final output $z_p(i,j)$ is obtained from the multiplexer (MUX) based on the value of horizontal index j.

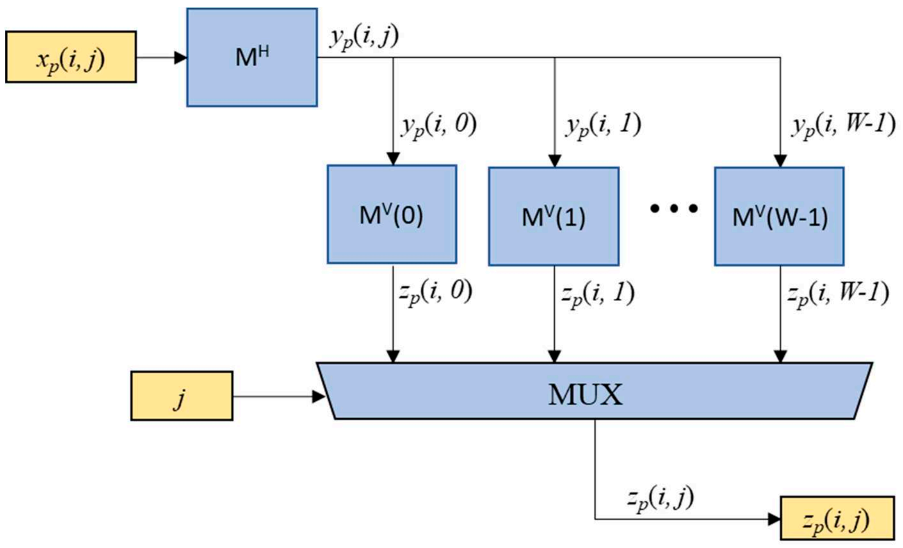


Figure 6. Block diagram for the two-dimensional RTB-MAXP engine.

The proposed max-pooling engine can be efficiently implemented by sharing the common components. Firstly, the MS in the R_v block and the D_v block for each v can be shared. Since the output values $c_v[0]$ and $c_v[1]$ of these two MSs are identical, they can be used commonly for a one-dimensional max-pooling engine. Secondly, since every vertical max-pooling engine $M^V(j)$ operates at non-overlapped timing for sequentially incoming $y_p(i,j)$ values, all components of $M^V(j)$ for $0 \le j \le W\text{-}1$ can be shared, except for registers used to store ranking-counting values and delay counting values.

4. Cascaded Maximum Based Max-Pooling *(CMB-MAXP)* Engine

In this section, another two-step max-pooling operation engine, named CMB-MAXP engine, is presented. In the CMB-MAXP engine, the horizontal axis max-pooling operation of (2) is accomplished through the cascaded maximum operations as illustrated in Figure 7. The cascaded F/Fs and maximum operators find the maximum values $y^w{}_p(i, j)$ for $1 \le w \le K\text{-}1$ of the input sequence elements $x_p(i, j)$, $x_p(i, j\text{-}1)$, $x_p(i, j\text{-}2)$, …, and $x_p(i, j\text{-}w)$. The cascaded maximum operation can be expressed in a recursive form, as shown in (4).

$$y_p^w(i,j) = \begin{cases} \max[x_p(i,j), x_p(i,j-1)] & ,w = 1 \\ \max[y_p^{w-1}(i,j), x_p(i,j-w)] & ,w \ge 2 \end{cases} \quad (4)$$

The $y^{k\text{-}1}{}_p(i, j)$ is chosen as the output sequence elements $y_p(i, j)$ of the horizontal axis max-pooling operation by a MUX according to the scalable kernel size parameter k, i.e. $y_p(i,j) = y_p^{k-1}(i,j)$.

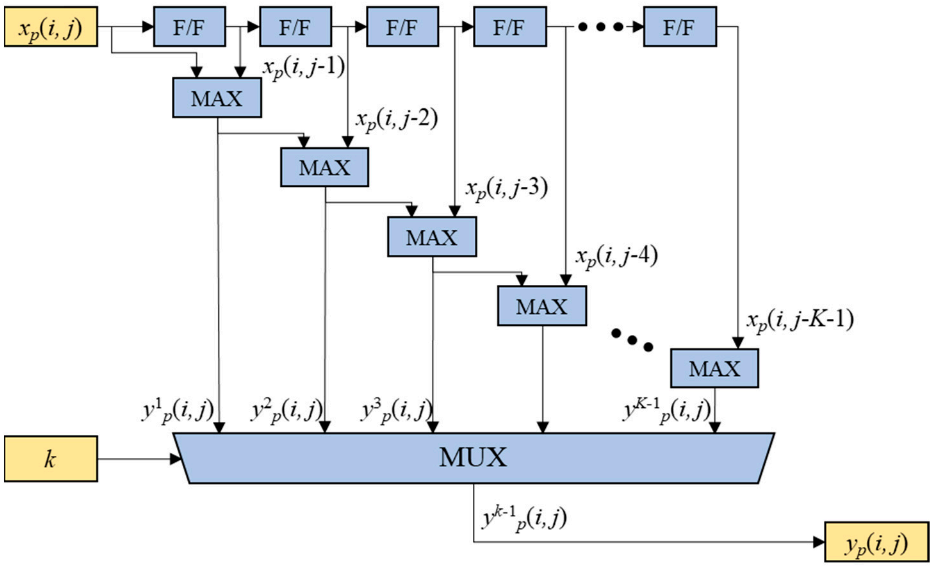
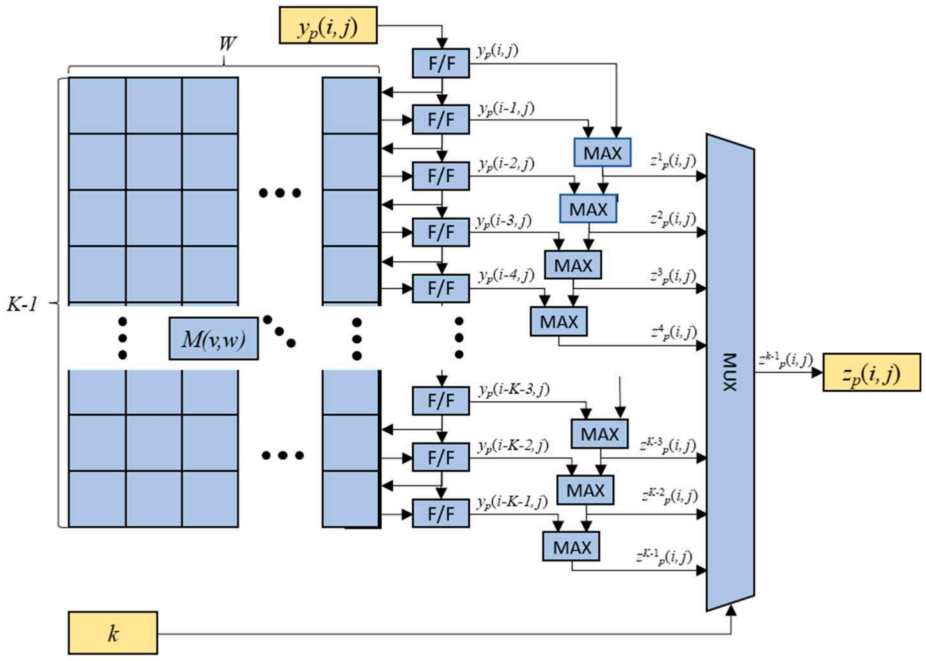


Figure 7. The horizontal axis max-pooling operation of the CMB-MAXP engine with the scalable kernel size *k*.

The structure of the vertical axis max-pooling operation is shown in Figure 8, and is almost identical to that of the horizontal axis max-pooling engine except for the additional usage of the two-dimensional memory elements $M(v, w)$ for $0 \le v \le K\text{-}1$ and $0 \le w \le W\text{-}1$. The memory elements are employed for restoring the previous row's values of the feature map $y_p(i\text{-}1, j)$, $y_p(i\text{-}2, j)$, …, $y_p(i\text{-}K\text{-}1, j)$ for the vertical max-pooling operations. The previously loaded elements $y_p(i\text{-}1, j)$, $y_p(i\text{-}2, j)$, …, $y_p(i\text{-}K\text{-}1, j)$ are fed into F/Fs from the memory element $M_p(0, j)$, $M_p(1, j)$, …, $M_p(K\text{-}1, j)$, and the new delayed element $y_p(i, j)$, $y_p(i\text{-}1, j)$,…, $y_p(i\text{-}K\text{-}2, j)$ are recursively fed into the corresponding memory elements. The maximum values $z^w{}_p(i, j)$ of the elements $y_p(i, j)$, $y_p(i\text{-}1, j)$, $y_p(i\text{-}2, j)$, …, $y_p(i\text{-}w, j)$ are obtained by comparators and the final output sequence element $z_p(i, j)$ is obtained from the MUX according to the value of *k*.



$y_p(i,j)$
W
K-1
M(v,w)
F/F
MAX
$y_p(i,j)$
$y_p(i\text{-}1,j)$
$y_p(i\text{-}2,j)$
$y_p(i\text{-}3,j)$
$y_p(i\text{-}4,j)$
$y_p(i\text{-}K\text{-}3,j)$
$y_p(i\text{-}K\text{-}2,j)$
$y_p(i\text{-}K\text{-}1,j)$
$z^1{}_p(i,j)$
$z^2{}_p(i,j)$
$z^3{}_p(i,j)$
$z^4{}_p(i,j)$
$z^{K\text{-}3}{}_p(i,j)$
$z^{K\text{-}2}{}_p(i,j)$
$z^{K\text{-}1}{}_p(i,j)$
MUX
$z^{k\text{-}1}{}_p(i,j)$
$z_p(i,j)$
k

Figure 8. The vertical axis max-pooling operation of the CMB-MAXP engine with scalable kernel size *k*.

5. Implementations

The proposed RTB-MAXP engine and CMB-MAXP engine were implemented for being employed in an FPGA-based CNN accelerator. The target model of the CNN was YOLOv4-CSP-S-Leaky which was developed for the object detection [9]. It consists of 108 layers including the 3x3 convolution layers, the 1x1 convolution layers, the residual addition layers, the concatenation layers, the max-pooling layers, and the up-sampling layers. The max-pooling layers are utilized for the SPP operation. When the input feature map size of the model is 256x256, the input feature map size of the max-pooling operations is 32x32 and the window sizes are 5, 9, and 13. Thus, the possible maximum window size *K* was chosen as 13 and the possible maximum width of the feature map *W* was set as 32 for the proposed max-pooling engines.

The max-pooling engines were designed using VHSIC Hardware Description Language (VHDL) and their behaviors were verified by simulations using the ModelSim which is the Mentor Graphics' simulation and debugging tool for digital logics. Figures 9 and 10 are the simulation results for the RTB-MAXP engine and the CMB-MAXP engine, respectively. The parameters *k* and *W* were named as ksize_i, width_i, and set to 5 and 0x20(=32). Note that prefixes of the signal are included in the signal of the figure. The signals inpchan_i, row_i, and valid_i were used for data synchronization with a start point of a new input channel, a start point of a new row, a valid point of data, respectively. The signal data_i indicates the data of a feature map aligned with the data synchronization signals. The output data signal data_o of the max-pooling engines comes out along with the data synchronization signals, i.e., outchan_o, valid_o.

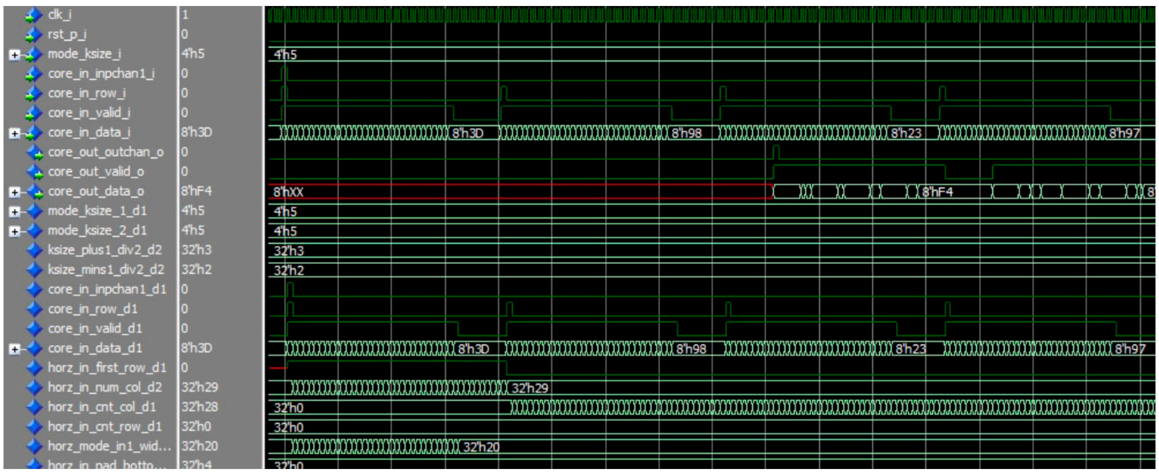


Figure 9. The simulation result of the RTB-MAXP engine with *k*=5 and *W*=32.

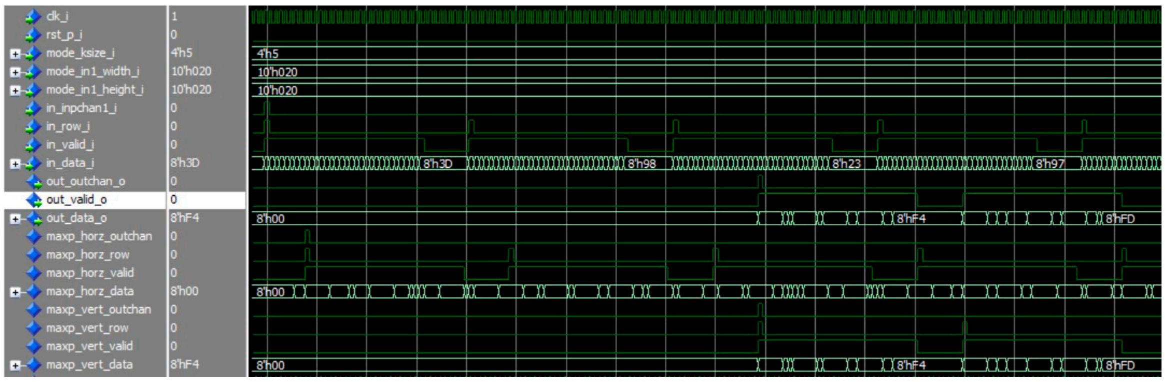


Figure 10. The simulation result of the CMB-MAXP engine with *k*=5 and *W*=32.

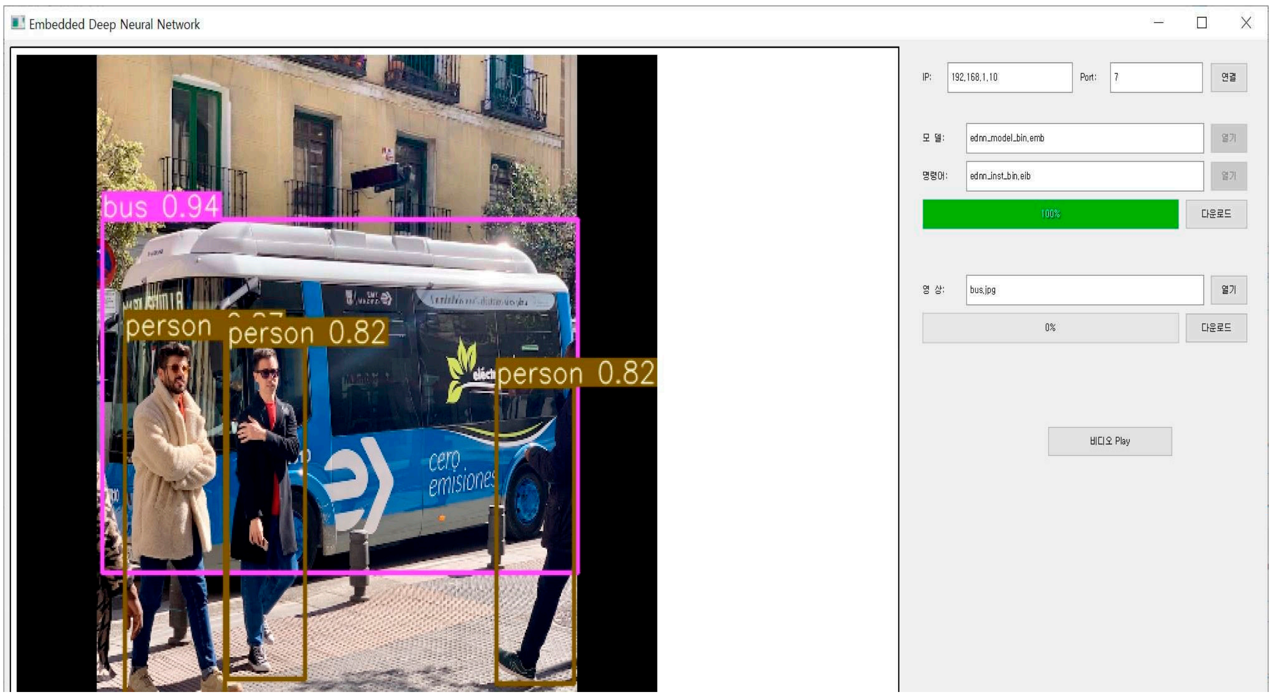
In order to accelerate the max-pooling operations for the CNN, either the 16 RTB-MAX engines or the 16 CMB-MAXP engines were used to process 16 feature maps ($0 \leq p \leq 15$) in parallel. These 16 max-pooling engines can increase the processing speed by 16 times in sacrifice of the FPGA resource. The designed two max-pooling engines were synthesized, placed, and routed in Xilinx Vivado Design Suite for the Xilinx VCU118 evaluation platform. The platform includes the Xilinx Vertex UltraScale+ FPGA XCVU9P. Table 2 shows the resource utilization of the RTB-MAXP engine and CMB-MAXP engine with K=13 and W=32 as a result of the implementation for the targeted FPGA. The RTB-MAXP engine required 157,515 LUTs and 99,342 FFs, that is, 13.4% and 4.2% of the total available resources, respectively. On the other hand, the CMB-MAXP engine required 28,765 LUTs, 2,688 LUTRAMs, and 76,906 FFs, that is, 2.43%, 0.45%, and 3.25%, respectively. The CMB-MAXP engine needs much fewer LUTs than the RTM-MAXP engine, but additionally requires few LUTRAMs. It is because of the buffer for the previous row data. The LUTs were used for this buffer in the RTM-MAXP engine while the LUTRAMs were used for that in the CMB-MAXP engine.

Table 2. Resource utilization of the RTB-MAXP engine and the CMB-MAXP engine with K=13 and W=32 implemented in Xilinx Vertex UltraScale+ FPGA XCVU9P.

	RTB-MAXP		**CMB-MAXP**	
	UA	UP	UA	UP
LUT	158,515	13.4%	28,765	2.43%
LUTRAM	-	-	2,688	0.45%
FF	99,342	4.2%	76,906	3.25%

UA: utilization amount, UP: utilization percentage.

The proposed RTB-MAX engine and CMB-MAXP engines were imported in our CNN accelerator. The CNN accelerator was tested on the target VCU118 evaluation platform board. Figure 11 shows the test results of the CNN using YOLOv4-CSP-S-Leaky model for the object detection [9]. It shows that the CNN accelerator detects three persons and one bus. Note that the comparison in terms of Mean Average Precision (mAP) is not included in this paper since the max-pooling operation is not the loss operation.



IP: 192.168.1.10 Port: 7
모 델: ednn_model_bin.emb
명령어: ednn_inst_bin.eib
100%
영 상: bus.jpg
0%

Figure 11. The test results of our CNN accelerator employing the CMB-MAXP engine on the VCU118 platform.

6. Conclusions

In this paper, we proposed two two-stage max-pooling engines for the max-pooling operation in CNN, i.e., the RTB-MAXP engine and the CMB-MAXP engine. The RTB-MAXP engine finds the maximum value by tracking the rank of the values within the window and the CMB-MAXP engine obtains the maximum value by cascading the maximum operations. They were implemented using VHDL and verified by simulations. They have been employed for and tested in our CNN accelerator targeting at the CNN model YOLOv4-CSP-S-Leaky for object detection.

References

1. Zhao, Z.; Zheng, P.; Xu, S.; Wu. X. Object detection with deep learning: A review. *IEEE trans. neural networks and learning systems* 2019, 30(11), 3212-3232.
2. Lee, D.-H. Fully Convolutional Single-Crop Siamese Networks for Real-Time Visual Object Tracking. *Electronics* 2019, 8, 1084. doi: 10.3390/electronics8101084.
3. Shawahna, A.; Sait S.; El-Maleh, A. FPGA-based Accelerators of Deep Learning Networks for Learning and Classification: A Review, *IEEE Access* 2018, 4, 1-41.
4. Huang, J.; Liu, X.; Guo, T.; Zhao, Z. A High-Performance FPGA-Based Depthwise Separable Convolution Accelerator. *Electronics* 2023, 12, 1571. doi: 10.3390/electronics12071571.
5. Xie, Y.; Majoros, T.; Oniga, S. FPGA-Based Hardware Accelerator on Portable Equipment for EEG Signal Patterns Recognition. *Electronics* 2022, 11, 2410. doi: 10.3390/electronics11152410.
6. Zhang, L.; Tang, X.; Hu, X.; Zhou, T.; Peng, Y. FPGA-Based BNN Architecture in Time Domain with Low Storage and Power Consumption. *Electronics* 2022, 11, 1421. doi: 10.3390/electronics11091421.
7. Lomas-Barrie, V.; Silva-Flores, R.; Neme, A.; Pena-Cabrera, M. A Multiview Recognition Method of Predefined Objects for Robot Assembly Using Deep Learning and Its Implementation on an FPGA. *Electronics* 2022, 11, 696. doi: 10.3390/electronics11050696
8. Zhou, H.; Xiao, Y.; Zheng, Z.; Yang, B. YOLOv2-tiny Target Detection System Based on FPGA Platform. ICBAIE 2022, 289-292.
9. Wang, C.; Bochkovskiy, A.; Liao. H. Scaled-yolov4: Scaling cross stage partial network. *Proceedings of the IEEE/cvf conf. comp. vis. and patt. recog.* 2021.
10. Bochkovskiy, A.; Wang, C.; Liao, H. Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934, 2020.
11. Rzaev, E.; Khanaev, A.; Amerikanov, A. Neural Network for Real-Time Object Detection on FPGA. *ICIEAM* 2021, pp. 719-723.
12. Archana, V. An FPGA-Based Computation-Efficient Convolutional Neural Network Accelerator. *IPRECON* 2022, Kollam, India, pp. 1-4, doi: 10.1109/IPRECON55716.2022.10059556.
13. Wang Z.; Xu, K.; Wu, S.; Liu L.; Wang, D. Sparse-YOLO: Hardware/Software Co-Design of an FPGA Accelerator for YOLOv2. *IEEE Access*, 8, pp. 116569-116585, doi: 10.1109/ACCESS.2020.3004198.
14. Zhao, B.; Chong Y.; Do, A.; Area and Energy Efficient 2D Max-Pooling for Convolutional Neural Network Hardware Accelerator. *IECON* 2020, pp. 423-427.
15. Zhao, D. F-CNN: An FPGA-based framework for training Convolutional Neural Networks *IEEE ASAP* 2016, London, UK, 107-114, doi: 10.1109/ASAP.2016.7760779