electronics

MDPI

Article

Design of a Generic Dynamically Reconfigurable Convolutional Neural Network Accelerator with Optimal Balance

Haoran Tong [1], Ke Han [1], Si Han [2,*] and Yingqi Luo [1]

[1] School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China; 767377325@bupt.edu.cn (H.T.); hanke@bupt.edu.cn (K.H.); luo_yingqi@bupt.cn (Y.L.)
[2] School of Information Management for Laws, China University for Political Science and Law, Beijing 100091, China
* Correspondence: hansi@cupl.edu.cn

Abstract: In many scenarios, edge devices perform computations for applications such as target detection and tracking, multimodal sensor fusion, low-light image enhancement, and image segmentation. There is an increasing trend of deploying and running multiple different network models on one hardware platform, but there is a lack of generic acceleration architectures that support standard convolution (CONV), depthwise separable CONV, and deconvolution (DeCONV) layers in such complex scenarios. In response, this paper proposes a more versatile dynamically reconfigurable CNN accelerator with a highly unified computing scheme. The proposed design, which is compatible with standard CNNs, lightweight CNNs, and CNNs with DeCONV layers, further improves the resource utilization and reduces the gap of efficiency when deploying different models. Thus, the hardware balance during the alternating execution of multiple models is enhanced. Compared to a state-of-the-art CNN accelerator, Xilinx DPU B4096, our optimized architecture achieves resource utilization improvements of 1.08× for VGG16 and 1.77× for MobileNetV1 in inference tasks on the Xilinx ZCU102 platform. The resource utilization and efficiency degradation between these two models are reduced to 59.6% and 63.7%, respectively. Furthermore, the proposed architecture can properly run DeCONV layers and demonstrates good performance.

Keywords: hardware accelerator; convolutional neural network; depthwise separable convolution; deconvolution; dynamically reconfigurable; on-chip computing scheme; resource utilization; high balance

Citation: Tong, H.; Han, K.; Han, S.; Luo, Y. Design of a Generic Dynamically Reconfigurable Convolutional Neural Network Accelerator with Optimal Balance. *Electronics* **2024**, *13*, 761. https://doi.org/10.3390/electronics13040761

Academic Editor: Costas Psychalinos

Received: 7 January 2024
Revised: 9 February 2024
Accepted: 10 February 2024
Published: 14 February 2024

1. Introduction

Nowadays, with the rapid development of technologies such as autonomous driving [1] and the Advanced Driver Assistance System (ADAS), there is a high demand for real-time performance in computer vision applications. Cloud computing is unable to meet the low-latency requirements, necessitating inference computations at the edge or on terminal devices. To meet diverse application requirements, an increasing number of new models are being iterated and proposed. Traditional convolutional neural networks (CNNs), while achieving high accuracy through complex convolution (CONV) layers, are resource-intensive and consume significant energy. This makes them challenging to be deployed on resource-constrained edge platforms. As a result, lightweight networks designed for edge and terminal devices, such as MobileNet [2,3], have been proposed. These networks leverage depthwise separable CONV layers to significantly reduce the system burden in scenarios where extremely high accuracy is not required. On the other hand, standard CNNs lose spatial information during the iterative computation process. However, deconvolutional neural networks (DCNNs) [4,5], such as FCN [6] and GAN [7], preserve the spatial information of the image. They can maintain the same resolution as the original image in the output and perform pixel-level classification of objects in the image.

The use of multiple types of CNN models in application scenarios has become a major trend in future development. For example, in applications such as pedestrian and vehicle

recognition and tracking, as well as multimodal sensor fusion [8], it is necessary to simultaneously or sequentially run multiple network models of different scales. Applications like image semantic segmentation and low-light image enhancement [9] utilize network models that include deconvolution (DeCONV) layers. This significantly increases the demand for a single hardware system to handle multiple computational tasks. Switching between different network models leads to significant fluctuations in the system workload. This is due to the differences in runtime and resource utilization. Fixed hardware architectures are prone to redundancy or insufficient performance, resulting in bottlenecks with low acceleration efficiency for certain algorithms. Additionally, the alternating deployment of network models may involve unconventional operators such as depthwise separable CONV and DeCONV. Given the complexity of current application scenarios, the key lies in improving the adaptability of CNN accelerators to dynamically changing applications.

Some excellent designs of CNN accelerators, such as Eyeriss [10–12] and TPU [13], have been efficient enough to accelerate standard CNN networks. However, they are not the optimal designs for lightweight networks. This is because lightweight CNNs have significant differences in data volumes and computation patterns compared to traditional CNNs. Therefore, it is necessary to make specialized and efficient improvements to the acceleration architectures for lightweight CNNs. Su et al. [14], Wu et al. [15], Zhao et al. [16], Xie et al. [17], Bai et al. [18], and Yu et al. [19] have gradually improved the frame rates of MobileNet inference tasks from 127.4 FPS to 325.7 FPS via implementing FPGA-based dedicated accelerators for lightweight CNNs. These studies focus on developing specialized accelerators and on-chip dataflows for lightweight CNNs, primarily targeting the unique computation pattern of depthwise separable CONV layers. However, for CONV computations, these dedicated designs lack parallel dimensions between channels, so they are significantly less efficient and performant. Additionally, accelerators specifically targeting the DeCONV layers of DCNNs are scarce. Zhang et al. [20] proposed the design of a DeCONV accelerator using reverse looping and stride hole skipping methods, and Yan et al. [21] introduced input-oriented mapping (IOM) for DeCONV, both to eliminate redundancy in memory access and zero value calculations. However, these specialized computation patterns inevitably require additional compute units, buffers, and specific hardware resources. This makes them unsuitable for general DCNNs and increasing hardware overheads and control system complexity. Another research study [22] proposed a method called transforming deconvolution into convolution (TDC), which partially mitigates redundancy issues by designing a load balancing scheme. However, the computational imbalance and space for the improvement of performance still exist.

In conclusion, existing research has shown that accelerator designs optimized for CONV layers often cannot efficiently handle depthwise separable CONV layers. Secondly, lightweight CNN accelerator designs are typically focused solely on processing depthwise separable CONV layers. Furthermore, DeCONV layers either require additional hardware resources or suffer from low resource balance. State-of-the-art commercial accelerator designs like the Xilinx DPU [23] exhibit significant differences in energy efficiency and resource utilization between the CONV layers and depthwise separable CONV layers. They often need additional hardware units for depthwise separable CONV layers. Current complex applications require the deployment of multiple models to address various needs. However, there is a lack of general-purpose accelerator architectures that can simultaneously handle standard CNN models, lightweight CNN models, and models involving DeCONV layers. To address this issue and improve the compatibility and energy efficiency balance between different models during edge inference hardware system deployment, this paper proposes a dynamically reconfigurable CNN accelerator architecture with high balance in complex applications. The work in this paper is divided into the following aspects:

1. Based on the prevalent processing element (PE) array structure in CNN accelerators, a versatile dynamic reconfigurable FPGA accelerator hardware architecture in high resource reuse is designed. A tiling computation flow that is highly compatible with the architecture is also proposed.

2. Considering the computational characteristics of depthwise separable CONV computation, our design dynamically configures the mapping of the computation in sub-arrays of PEs and optimize the on-chip dataflow for depthwise separable CONV. This allows the proposed computing engine to alternate between depthwise CONV and pointwise CONV while minimizing the redundant memory access operations between the two computations. The proposed accelerator hardware architecture achieves compatibility with depthwise separable CONV layers and exhibits higher resource utilization.
3. Considering the characteristics of DeCONV computation, this design splits the kernels into parts and maps them to sub-arrays of PEs for separate computation. This transforms a DeCONV computation into multiple CONV computations without zero-padding, enabling generalization between the CONV layers and DeCONV layers in terms of computational resources. This eliminates redundant zero-value multiplies and accumulates as well as unnecessary storage operations, thereby achieving compatibility with DeCONV layers and improving resource utilization.

2. Background and Analysis of CNN Acceleration

2.1. Standard CONV and Acceleration Design

2.1.1. On-Chip Parallel Computation for CONV

The key to a neural network accelerator lies in maximizing the parallel computing capability of the hardware platform to improve the parallelism during CONV computations. CONV layers are arranged sequentially. Computations within a single CONV layer are shown in Figure 1. Due to the corresponding relationship between the kernels and input feature maps, the continuously looping computations across dimensions can be fully parallelized for high acceleration efficiency at the cost of hardware resources.

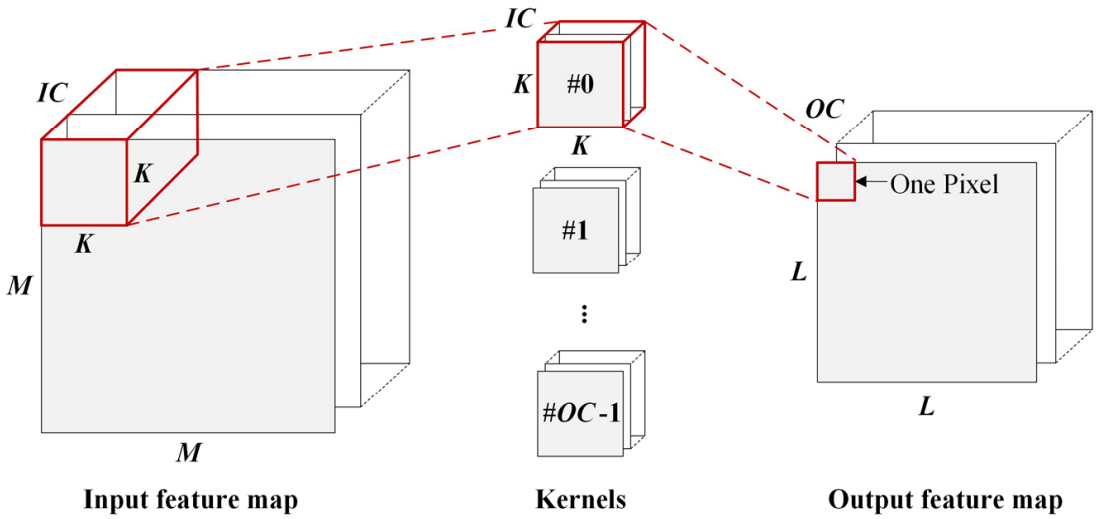

Figure 1. Computations of a CONV layer.

Therefore, a major focus in accelerator design is striking a balance between hardware resources and acceleration performance. Parallel dimensions depend on the computing mode of the CNN model. Parallel computations mainly involve the following three dimensions:

1. Sliding window parallelism: Within the sliding window corresponding to a kernel, $K \times K$ multiplications can be executed in parallel. Each channel of a kernel uses $K \times K$ multipliers, enabling the calculation of one sliding window in a single cycle for that channel. This means that by increasing the number of multipliers used to $K \times K$ times, the computation time can be reduced to $1/(K \times K)$ of the original time. The sliding window parallelism strategy is suitable for networks with relatively uniform kernel sizes.

2. Input channel parallelism: Since the convolutions of different channels in the input feature map are independent, the pixels at the same position in different channels can be convolved in the same cycle. When the number of parallel channels is C, the computation time becomes $1/C$ of the original time.
3. Output channel parallelism: The number of output feature map channels in a CONV layer depends on the number of kernels. Each channel of the input feature map is convolved with the corresponding channel of each kernel, and the feature data are reused. Performing convolution with multiple kernels simultaneously reduces the number of data reads and improves the throughput of the computational units. This parallelization method has the highest requirement for computational and buffer resources.

2.1.2. Conventional Accelerator Design for CONV

The mainstream hardware acceleration method for CONV utilizes a PE array, which is composed of interconnected PE units, to perform CONV computations. Each PE is only responsible for one or some multiply–accumulate (MAC) operations. After transferring data between regularly arranged PEs, CONV can be completed by the PE array. This design also can achieve a very high degree of parallelism, because the PEs are computing the MACs simultaneously. The separate MAC unit design is currently used in various FPGA designs, such as CNN accelerators [10,13], and hardware implementation of other algorithms [24].

The row stationary (RS) dataflow [10] introduced by Eyeriss is efficient for CONV. The computing engine of Eyeriss, which is a PE array consisting of a 12×14 grid of PEs, can be divided into multiple sub-arrays to map computations of different parallelisms. In the RS dataflow, the movement of input features, weights, and MAC results in the PE array taking three different approaches. As shown in Figure 2, the weight values are sent to each PE using a horizontal mapping approach. PEs on the same row simultaneously hold the same weight values, mapping a row of a kernel. The input feature values are sent to each PE using a diagonal mapping approach. PEs on the same diagonal line simultaneously hold the same feature values, mapping a row of feature elements. The partial sums (psums) in a sliding window are vertically accumulated. This dataflow concentrates data reuse and accumulation in the PE array. It reduces power consumption and time consumption by reducing burst data access to the buffer and off-chip DRAM.

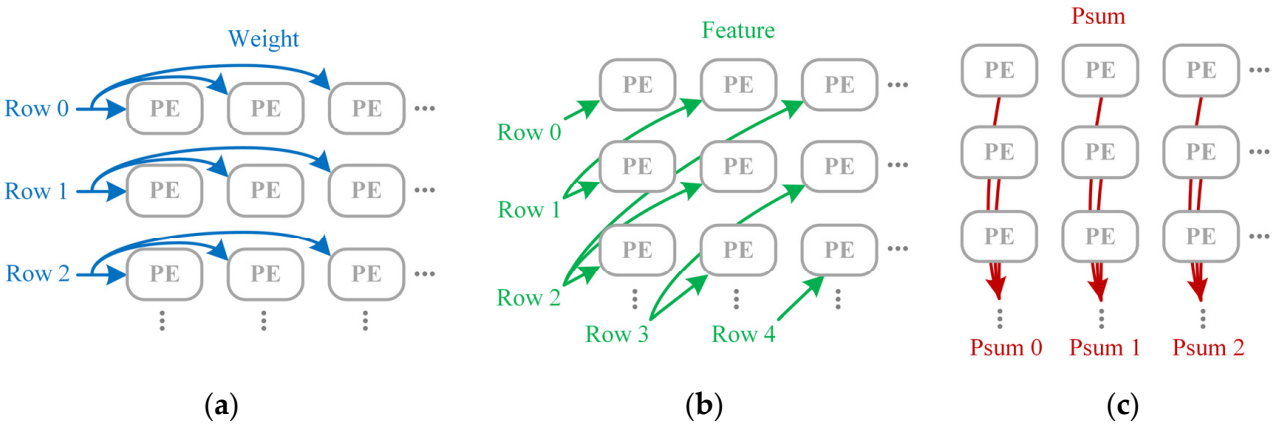

Figure 2. RS dataflow in a PE array: (**a**) dataflow of weights; (**b**) dataflow of features; and (**c**) dataflow of psums.

2.2. Depthwise Separable CONV and Challenges in Acceleration

Depthwise separable CONV splits the CONV process into two steps: depthwise CONV and pointwise CONV. By separating the accumulation within sliding windows and the accumulation across channels, it significantly reduces the number of parameters and computational complexity. The characteristics of depthwise CONV involve using a single kernel with the same number of channels as the input feature map to independently convolve

each channel, resulting in an output with the same number of channels. Pointwise CONV mainly utilizes multiple 1×1 kernels to compute the results obtained from depthwise CONV in the same manner as standard CONV. Although depthwise separable CONV has lower computational requirements, its efficiency is often lower in hardware accelerators.

There are two main reasons for this. Firstly, depthwise convolution has low computational intensity, leading to more time spent on memory access in hardware implementation. Two performance evaluation metrics need to be considered in the network model: computational load and memory access load. The computational load (L_{comp}) represents the number of MAC operations required during the forward propagation of the network. The majority of the computational load in CNNs is concentrated in the CONV layers. The computational load of a CONV layer can be calculated as follows:

$$L_{comp} = M^2 \cdot K^2 \cdot IC \cdot OC, \tag{1}$$

The memory access load (L_{mem}) represents the amount of data exchanged in the memory during inference calculations. Assuming that the on-chip buffer is large enough and each CONV layer operation only requires reading the memory once, the memory access load of a CONV layer can be calculated as follows:

$$L_{mem} = \left(K^2 \cdot IC \cdot OC + M^2 \cdot OC\right) \cdot 4, \tag{2}$$

The computational intensity (I) quantifies how many floating-point operations are performed per byte of memory exchange during the model's computation, as measured in FLOPs/Byte. It is a quantifiable indicator of the efficiency of the model's memory usage, which is calculated as follows:

$$I = L_{comp} / L_{mem}, \tag{3}$$

For standard CONV, taking VGG16 [25] as an example, the floating-point computational load for a single inference task is approximately 15 GFLOPs, and the memory access load is around 600 MB. Therefore, its computational intensity is 25 FLOPs/Byte. In the case of MobileNet, which uses depthwise separable CONV, the computational load is approximately 0.5 GFLOPs, and the memory access load is 74 MB. Compared to the memory access load, the computational load has been significantly reduced, resulting in a lower computational intensity of only 7 FLOPs/Byte. The low computational intensity leads to lower acceleration efficiency for MobileNet, making it suitable for hardware platforms with limited computational resources and low parallelism. It cannot fully utilize the computational resources of large-scale parallel computing platforms.

Secondly, due to the lack of output channel parallelism during depthwise CONV, deploying it in highly parallel accelerators can lead to a decrease in acceleration efficiency. When computing standard CONV on hardware accelerators, the input is typically fetched once and then reused by multiple kernels to minimize memory access. In lightweight networks, each output of depthwise CONV is the result of a single-channel MAC operation. Therefore, there is no parallel dimension among different kernels. Deploying lightweight networks on traditional accelerators results in a lack of computational parallelism, directly reducing efficiency. Taking the mapping of standard CONV on TPU [13] as an example, as shown in Figure 3, in the architecture based on the weight-stationary (WS) dataflow, each column of the PE array processes the computation of a single output channel (a single kernel). The weights are preloaded into the PEs, and the input feature map is loaded from the global buffer into the input buffer, which is then sent to all the columns in the PE array to perform convolutions with different kernels. Unlike standard CONV, which performs parallel computations with multiple columns of PEs, depthwise CONV only uses a single kernel. Therefore, there is only one column in the PE array needed to map the computation of the only kernel, while the other columns remain unused.

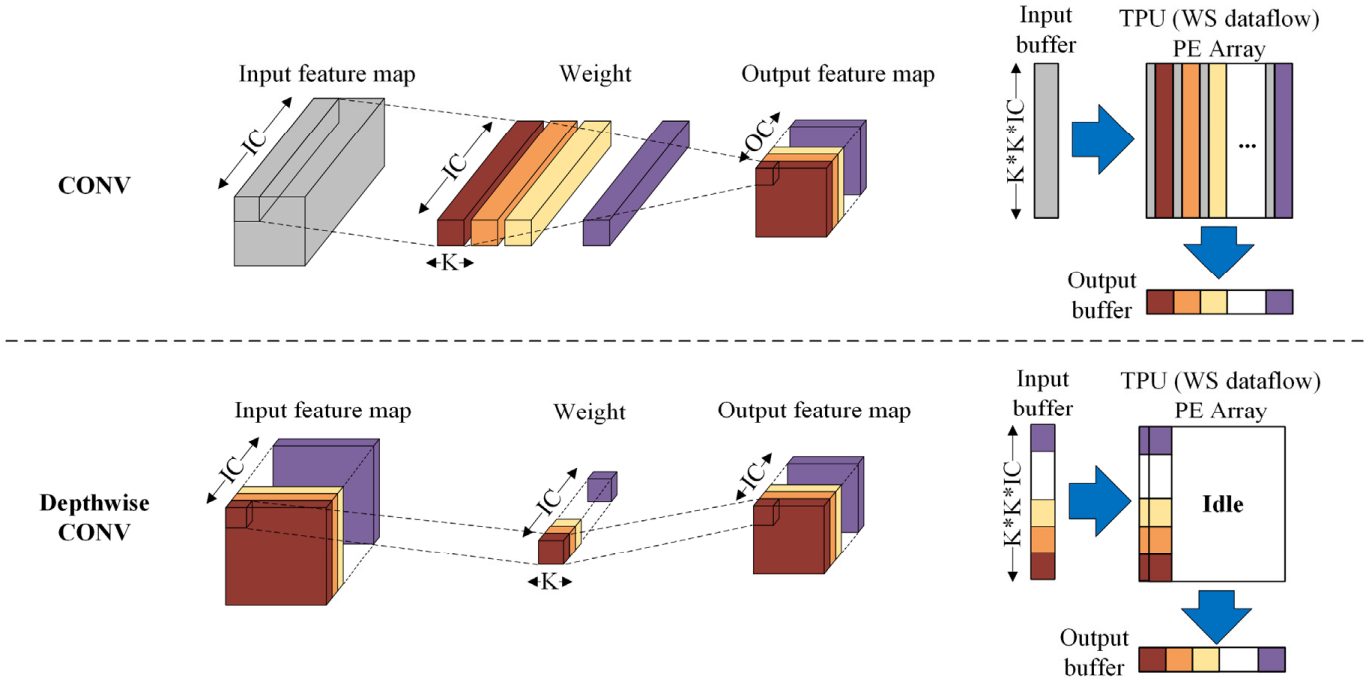

Figure 3. CONV and depthwise CONV in TPU.

2.3. DeCONV and Challenges in Acceleration

DeCONV is primarily used to increase the resolution of input images to obtain more detailed features. During computation, zero values are inserted around the boundaries of the feature map and between every two pixels. This allows convolution operations to generate feature maps larger than the original size. Consequently, when running the DeCONV layer, a significant portion of the accelerator's resources is allocated to performing MAC operations involving zero values. These unnecessary computations result in the wastage of hardware resources. Therefore, to avoid allocating resources to zero value computations, optimization of the dataflow of DeCONV is necessary.

The IOM computation scheme [21] evenly distributes the computation tasks of the input feature map pixels to each PE unit, while avoiding the process of padding the input feature map with zero values. The computation can be described as follows. (1) The weights of the kernel are flipped vertically and horizontally. (2) Each individual pixel of the input image is multiplied element-wise with the flipped 3×3 kernel. (3) The resulting 3×3 outputs are arranged as shown in Figure 4. The overlapping pixels are accumulated, while the edge pixels are discarded in the final output. (4) Repeat the above steps for all the input feature map pixels until the final output is obtained.

The challenge with this novel computation scheme lies in the need to design a specialized dataflow to accommodate the kernel-flipping operation, element-wise multiplications of a single feature pixel and the arrangement of the output image. It is difficult for accelerators designed for standard CONV and depthwise separable CONV to efficiently support such a different computation scheme. DeCONV has higher computational complexity compared to CONV, and designing a dedicated DeCONV computing engine can result in low resource utilization. In complex application scenarios, it is common to encounter situations involving both CONV layers and DeCONV layers within the same model or across different models. Hence, it is crucial to propose a DeCONV dataflow that is compatible with the CONV computation mode, without the need for a specialized DeCONV engine.

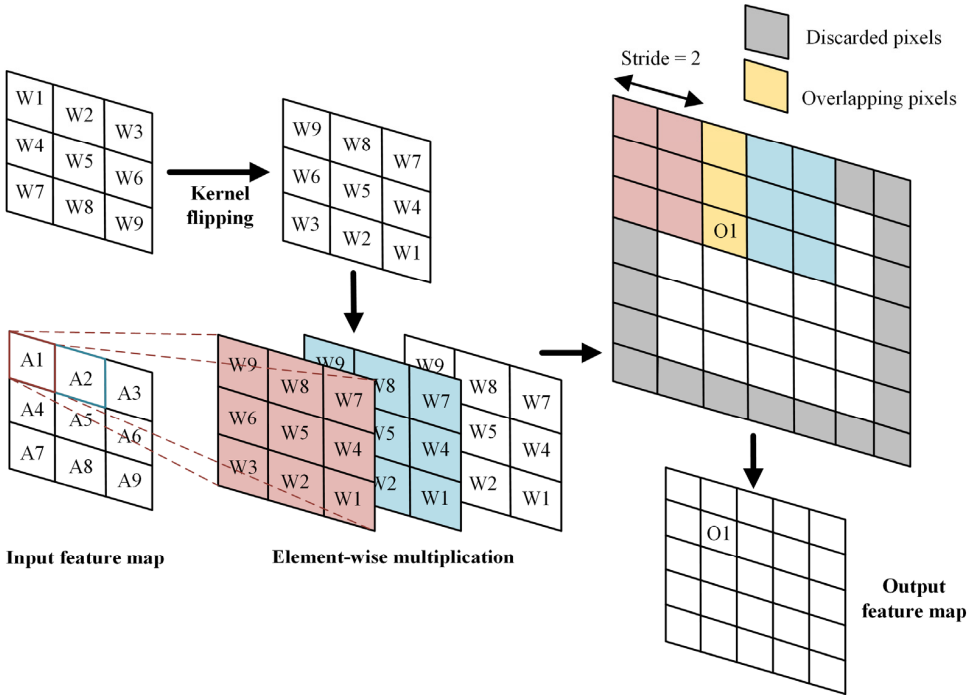

Figure 4. IOM computing scheme for DeCONV.

3. System Architecture of the Proposed Accelerator

3.1. Top-Level Architecture

The primary objective of the proposed design of the versatile dynamic reconfigurable accelerator is to dynamically adjust the hardware accelerator's computational capabilities based on the characteristics of different CONV layers while addressing the trade-off between resource utilization and efficiency. This paper proposes a resource-oriented tiling computation flow to maximize the computational parallelism. Specifically, data from the off-chip memory (DRAM) are divided into blocks and enter the on-chip buffer (BRAM), where the content is further divided and fed into the PE array for computation. The block size of the data is adapted to the size of the PE array. This design achieves parallelism between memory access and computation while reducing the amount of data transferred in each intra-chip communication. It also eliminates the memory access bandwidth bottleneck issue and reduces the size of the on-chip buffers. Meanwhile, in order to improve the balance between the accuracy and efficiency of the CNN inference tasks, quantized 8-bit fixed point (INT8) parameters instead of 32-bit floating point (FP32) parameters are used to compress the implemented CNN models. After quantization, the hardware resources for computation and memory units are further reduced with little degradation of accuracy. The details of the quantized models are shown in Section 5.

The tiling computation flow and top hardware modules are illustrated in Figure 5. All the weight data, input images, and intermediate output results are stored in the off-chip DRAM. The on-chip buffer consists of a total of 1164 KB of BRAM, and a fixed number of blocks of data are stored in it at a time. The data entering buffer are continuously reused and further divided into PEs. After a fixed computation cycle, the psums of the output are accumulated in the output buffer. Once the computation is completed, the resulting output feature map is returned to the corresponding address in DRAM as the input data for the next layer.

The hardware architecture of the designed accelerator in this paper is depicted in Figure 6. The accelerator comprises several key modules, including the on-chip buffer, control unit, address generate unit (AGU), data allocation unit (DAU), PE array, and post-process modules. The on-chip buffer consists of the input buffer (INF_BUF), weight buffer (OUTF_BUF), and output buffer. These buffers are respectively responsible for storing the

tiled input feature map data, weight data, and results from the computation units. The AGU generates addresses for reading and writing for the buffers. The DAU handles the transmission of 128-bit data to the corresponding registers in the PEs. The PE array is an array of interconnected PEs that perform MACs and data flowing. The output of the PE array is connected to an adder tree, which outputs the final convolutional results. The post-process units encompass operations such as quantization, dequantization, pooling, and ReLU, performing necessary computations beyond convolutions.

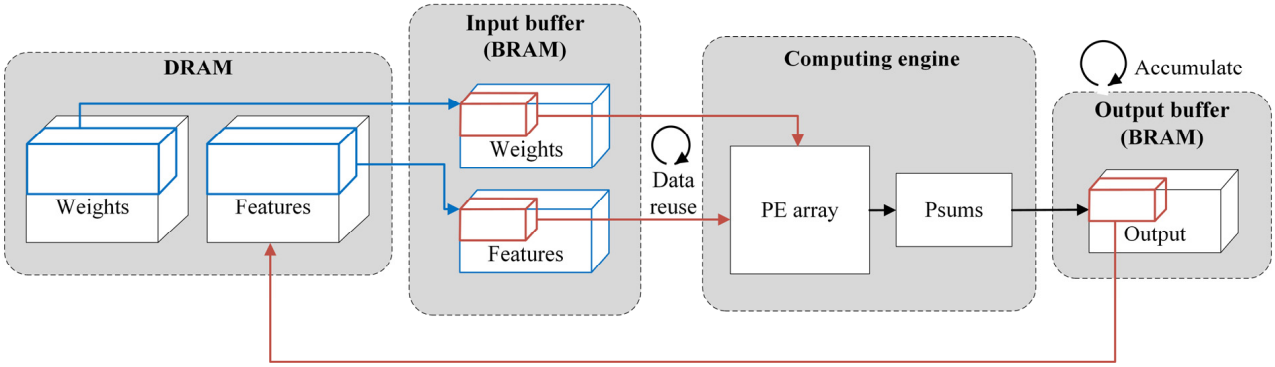

Figure 5. The proposed tiling computation flow and top modules.

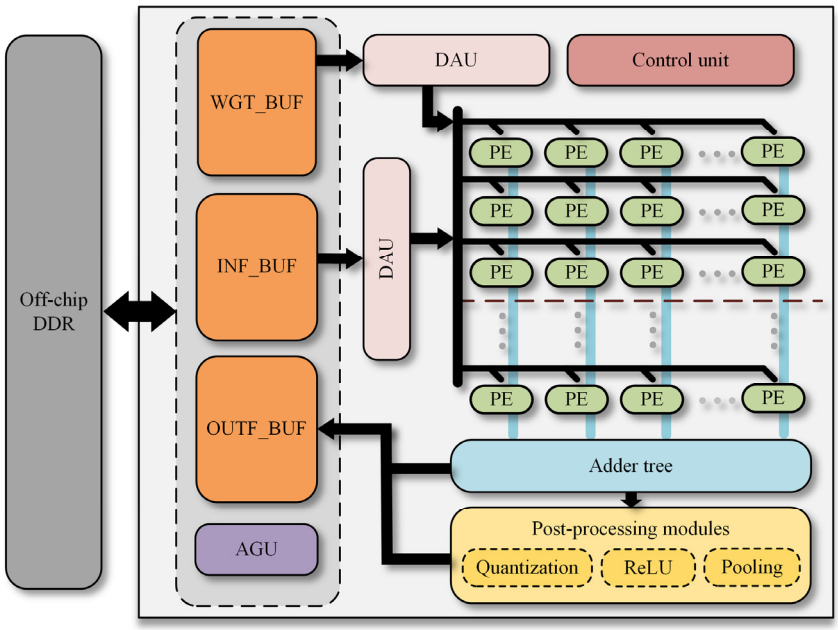

Figure 6. Top hardware architecture of the proposed accelerator.

3.2. Design of the PE Array and Dataflow

The designed computing engine is implemented using a PE array architecture. A strategy of sliding window parallelism and input channel parallelism is adopted for the mapping of the CONV operations. This is to avoid the lack of output channel parallelism, which leads to low resource utilization in computing depthwise CONV. This design decomposes and partitions the input feature map into the PE array for mapping along the channel, row, and pixel dimensions, achieving load-balancing and efficient on-chip dataflow with a simple control. For the efficiency of accelerating standard CONV, the design of the PE array size and the flow of different types of input data in the array in this paper refers to the idea of an RS dataflow. Based on this, a dynamically reconfigurable design for the internal structure of the PE unit and the data mapping method on the array is proposed.

Compared to Eyeriss, which focuses only on accelerating CONV computations, our solution can dynamically support multiple types of CONV.

In the 12 × 14 PE array, this design maps the computations of each sliding window to the PEs in the same column of a sub-array, with each PE responsible for calculating a row in the sliding window. Since most CONV layers in computer vision applications currently use small 3 × 3 kernels, three PEs are required to calculate one sliding window when each PE calculates one row. Therefore, the size of the sub-array is designed to be 3 rows. The 14 columns of PEs are designed to perform parallel MACs of 14 sliding windows per cycle. Four 3 × 14 sub-arrays parallelize the convolution calculations for four input channels. Each PE is responsible for calculating the MAC of one row (1 × 3 elements) within the sliding window. Therefore, each PE is internally designed with three sets of INT8 fixed-point MAC logic, as shown in Figure 7. By adding the results of the three PEs in the same column, the computation of one sliding window can be completed. Each PE contains three registers for temporary storage of weights and feature values. Since the kernel size is small, the WS dataflow can be used to map the weight values. The movement of the sliding window is achieved via the flow of feature values inside the PE. Specifically, the weights remain stationary in the internal registers, while the feature values are element-wise shifted in the registers, accomplishing the horizontal sliding of the kernel on the input feature map.

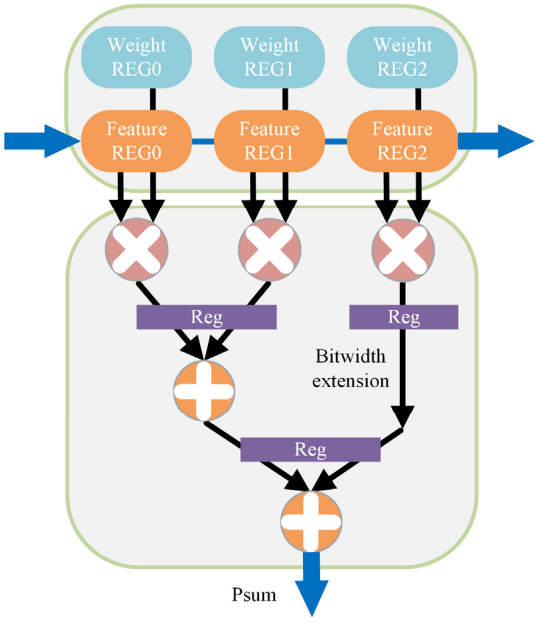

Figure 7. Design of the PEs. Each PE contains three groups of registers, three groups of multiplication and two stages of accumulation.

After the buffered data are tiled, they are distributed to various PEs through data buses. The designed dataflow in the PE array, which is shown in Figure 8, follows and optimizes the data movement of the RS dataflow. When a CONV layer is initialized, each row of the kernel is uniformly sent to the registers of all the PEs in the corresponding row and stored. The PEs in the same row hold the same weight values, allowing a complete kernel to be fixedly mapped to the three PEs in a column without the need for movement. The input feature values are sent to each PE array diagonally. As the sliding window moves horizontally, the feature registers inside a PE perform shifting and updating in each cycle. Therefore, within a single cycle, a 3 × 14 sub-array can perform the convolution computation between a 16 × 3 input feature matrix and a 3 × 3 kernel. In N cycles, the convolution computation for a 16 × N region can be completed (the initialization of weights and features requires 2-cycle shifting in the PE registers). After completing the internal computation, the PE sends the results to the adder tree at the bottom of each column to accumulate the results of the three rows in the sliding window.

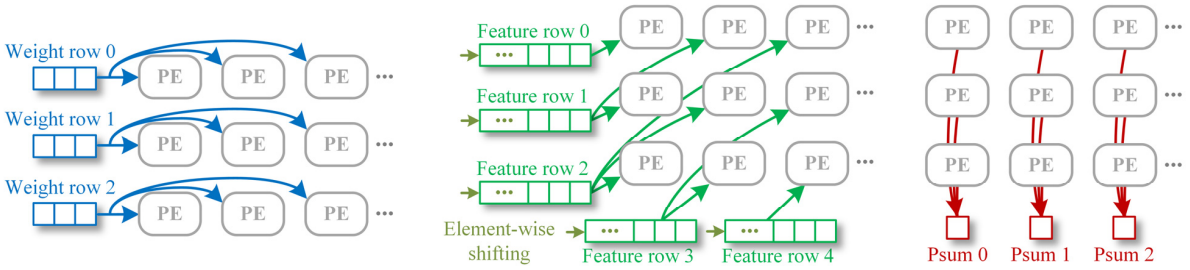

Figure 8. Optimized dataflow in the PE array: (**a**) dataflow of weights; (**b**) dataflow of features; and (**c**) dataflow of psums.

After the computation starts in each PE unit, each valid MAC result needs to be further accumulated across the rows and channels within the sliding window to obtain the output feature values. This is achieved through a two-stage adder tree structure that performs data accumulation within the columns of the PE array, as shown in Figure 9. The sub-array completes the accumulation of the MAC results for the three rows of the sliding window. The second stage of the adder tree completes the accumulation of results between different channels, computed by four sub-arrays. The accumulated data are then sent to output buffer and await further accumulation with results from other channels that are yet to be computed. The output buffer consists of 14 FIFOs with a depth of 512, and it receives and accumulates the corresponding psums from different sub-arrays. Since the data output from the array follows a sequential order, the use of FIFO efficiently reduces the cost of the BRAM resources.

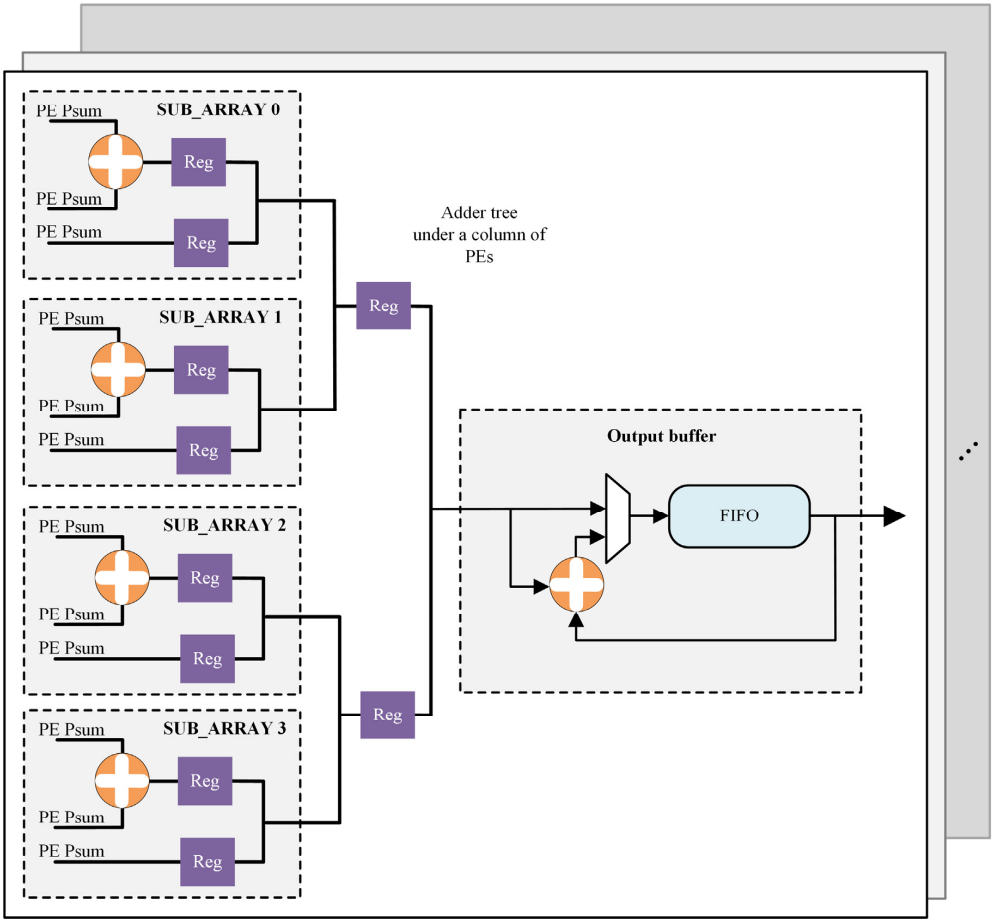

Figure 9. Adder trees and output buffers.

4. The Proposed Unified Computing Scheme

Based on the designed reconfigurable accelerator architecture mentioned above, the key to improving the balance and resource utilization among various CONVs lies in implementing on-chip computations with minimal heterogeneity. Therefore, in conjunction with the designed reconfigurable accelerator, our proposed computing strategy mainly focuses on maximizing the similarity among the three types of CONV operations to unify the dataflows and hardware designs.

4.1. The Applied Computing Scheme of the CONV

For the computation of standard CONVs, after the data are loaded into the on-chip buffer in block format, a further tiling strategy is adopted to input the data into the PE array for computation, as shown in Figure 10. The PE array can process a $16 \times 3 \times 4$ feature map convolved with a $3 \times 3 \times 4$ kernel per cycle. Therefore, the size of the input data block entering the array each time is $16 \times 3 \times 4$ for the input image block and $3 \times 3 \times 4$ for the weight block. The computation follows a sequential order in four dimensions, and the computation pattern for CONV is as follows:

1. Initially, the array will process the convolution between $16 \times M \times 4$ features and a kernel of 4 channels in pipeline, as indicated by arrow a in Figure 10. Due to the 2-cycle initialization for features and weights in PE registers, thus requiring M cycles.
2. After M cycles, the subsequent convolutions for the remaining channels are performed in a batch-wise manner, where each batch consists of 4 channels. This operation is repeated $C/4$ times, as indicated by arrow b.
3. The tiled convolution operation above needs to be repeated $(M-16)/14+1$ times to complete all the convolutions of a single kernel with the input feature map, as indicated by arrow c.
4. For a CONV layer with N kernels, the above 3-step tiling scheme needs to be repeated N times to obtain the final output map with a channel number of N.

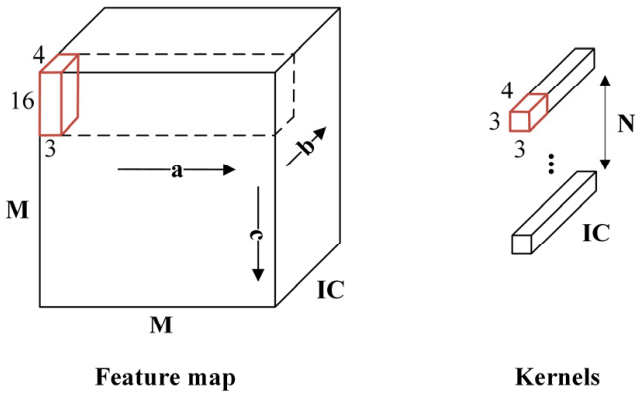

Figure 10. The computing strategy for the CONV.

Therefore, for a 3×3 CONV, the theoretical number of cycles required for the PE array to compute a single CONV layer is $M \times (IC/4) \times ((M-16)/14+1) \times N$.

4.2. The Unified Computing Scheme of the Depthwise Separable CONV

4.2.1. The Designed Computing Strategy of the Depthwise Separable CONV

Instead of writing data back to the off-chip DRAM after completing the depthwise CONV and then reading for the pointwise CONV, treating these two processes as separate layers, the intermediate data are stored in the on-chip BRAM in this proposed design. Due to the absence of parallel accumulation between channels in depthwise CONV and its immediate follow-up by pointwise CONV, additional data allocation logic and buffers are needed. For depthwise CONV, the designed architecture can satisfy the convolution operations for each channel. However, instead of accumulating the convolution results

of each sub-array in the adder tree, the results are stored in 12 buffers corresponding to 12 channels, each with a size of 14×1. The purpose of these buffers is to temporarily store the depthwise CONV outputs in 3 cycles (each cycle outputs 4 channels with 14 convolution results) as inputs for the following pointwise CONV.

During pointwise CONV, the buffered depthwise CONV results of 12 channels are used as input data. As shown in Figure 11, a data block of the size $14 \times 1 \times 12$ is sent to PE array. In this way, the amount of feature values for pointwise CONV is smaller than the total weight values of the point kernels. Therefore, the dataflow in the PE array can be changed to the input stationary (IS) dataflow. The $14 \times 1 \times 12$ data block of the feature values is stored in the corresponding PEs, and different point kernels are kernel-wise shifted in registers, maximizing the reuse of the depthwise CONV results.

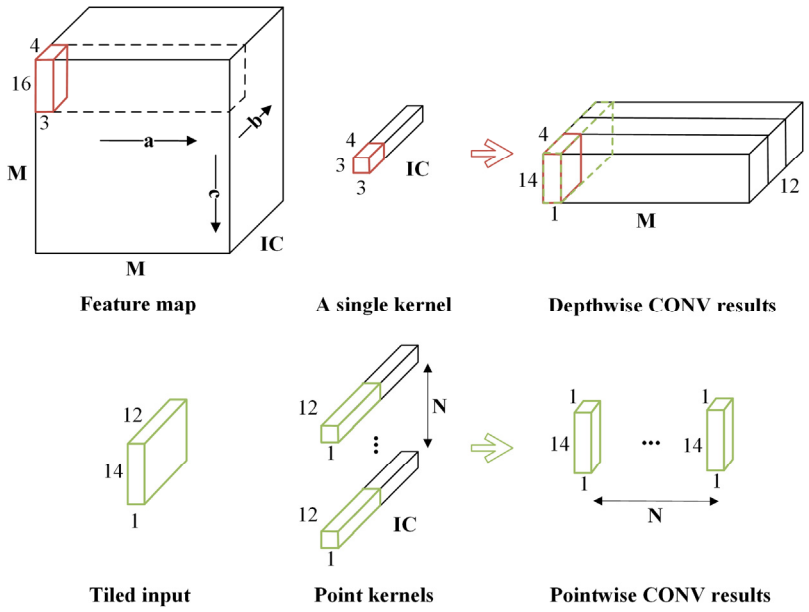

Figure 11. The computing strategy for the depthwise separable CONV.

Only after the pointwise CONV is completed, is the accumulation performed through an adder tree, and the results are stored in the output buffer. This design of the dataflow maximizes the utilization of the PE array resources for depthwise CONV and pointwise CONV, minimizing idle resources.

Existing generic accelerators require depthwise CONV and pointwise CONV to be performed on separate hardware modules, which inevitably results in additional resource and power consumption. Our design computes these two CONV steps in the same engine so that the resource utilization and power consumption are further optimized. This ensures computational efficiency, improves resource utilization, balances the computational load between these two CONVs, and performs seamless integration. It only requires a smaller intermediate data buffer space to connect the two CONVs, avoiding off-chip communication while efficiently utilizing on-chip BRAM resources.

4.2.2. Reconfigurations of the Pointwise CONV

During the pointwise CONV process of depthwise separable CONV, the kernel size becomes 1×1. Since the 1×1 sliding window does not involve the overlapping of feature elements during sliding, the feature values are not updated in a diagonal flow anymore. Instead, it is channel-wise mapped to each row of PEs, as shown in Figure 12, and the feature value is assigned to the corresponding column of PEs according to the row order of the input feature map.

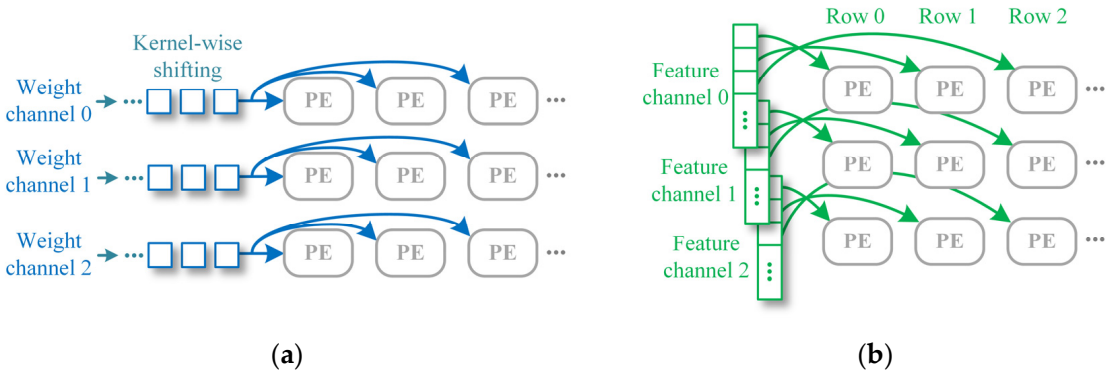

Figure 12. Dataflow of pointwise CONV in the PE array: (**a**) dataflow of weights; and (**b**) dataflow of features.

Parallel computations of three MACs are not required within the PE when computing pointwise CONV. Therefore, the other two groups of registers and MAC units inside a PE do not need to be enabled during pointwise CONV. Also, the data-shifting operation is transferred from the feature register to the weight register. The structure of the PEs is reconfigured as shown in Figure 13. When calculating depthwise separable CONV, a group of control signals will be generated to control the reconfiguration of the PEs and dataflow. The finite-state machine of computation will also be configured into two phases of depthwise CONV and pointwise CONV. Compared with standard CONV, the depthwise CONV phase removes the logic of accumulation in the adder tree. The results are directly stored to the middle buffer instead. The pointwise CONV phase changes the PEs into one set of MAC logic and the dataflow into an IS dataflow by control signals.

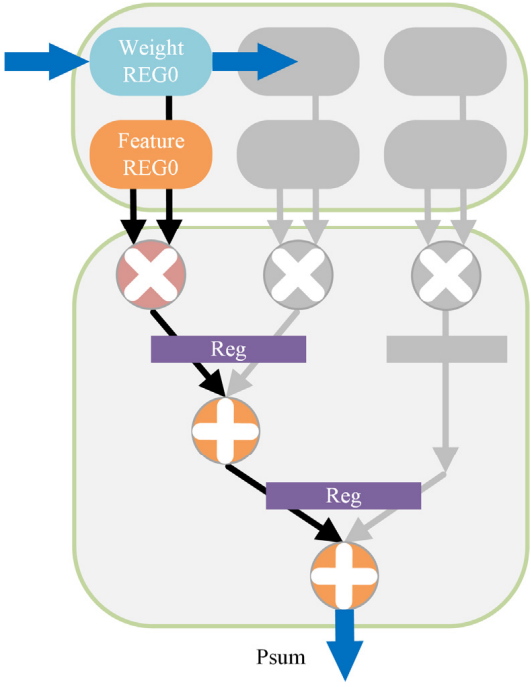

Figure 13. Reconfigured PEs for the pointwise CONV.

The process above is the reconfiguration strategy for the depthwise separable CONV. Through the phase switching of the control signal, it can maximize the utilization of power consumption to perform computations. By disabling unnecessary computing units, redundant power consumption can be avoided. Thus, the efficiency of acceleration is further improved.

4.3. The Unified Computing Scheme of the DeCONV

4.3.1. Optimization of the DeCONV

The computation of the DeCONV, after zero-padding, is consistent with CONV and can be adapted to traditional architectures. Therefore, the acceleration focus lies in handling redundant zero value computations. The positions where zero values are inserted into feature maps have a fixed pattern. Consequently, the valid MACs follow this pattern as well during convolutions in DeCONV. This property can be leveraged to optimize the dataflow of DeCONV. Drawing upon the TDC method, the proposed DeCONV method in this paper also avoids zero value computations and calculation overlaps by partitioning the original kernel into several sub-kernels. However, our method does not require the insertion of zero values into the sub-kernels to accommodate the computing engine instead. This is because our designed hardware architecture can dynamically adjust the internal registers of the PE, enable/disable MAC operations, and allocate data buffers to adapt to convolutions of different kernel sizes.

The designed DeCONV method is illustrated here using an example of 4×4 kernels. As shown in Figure 14, when the stride is 2, zero values are inserted around each element of the input feature map. Thus, when the kernel slides over the feature map for convolution, only four corresponding cases occur, as represented by the four colors in Figure 14. Accordingly, the kernel can be divided into four 2×2 sub-kernels, with each sub-kernel corresponding to a specific case. The positions of the sub-kernel elements after splitting are also distinguished by the corresponding colors in the original kernel.

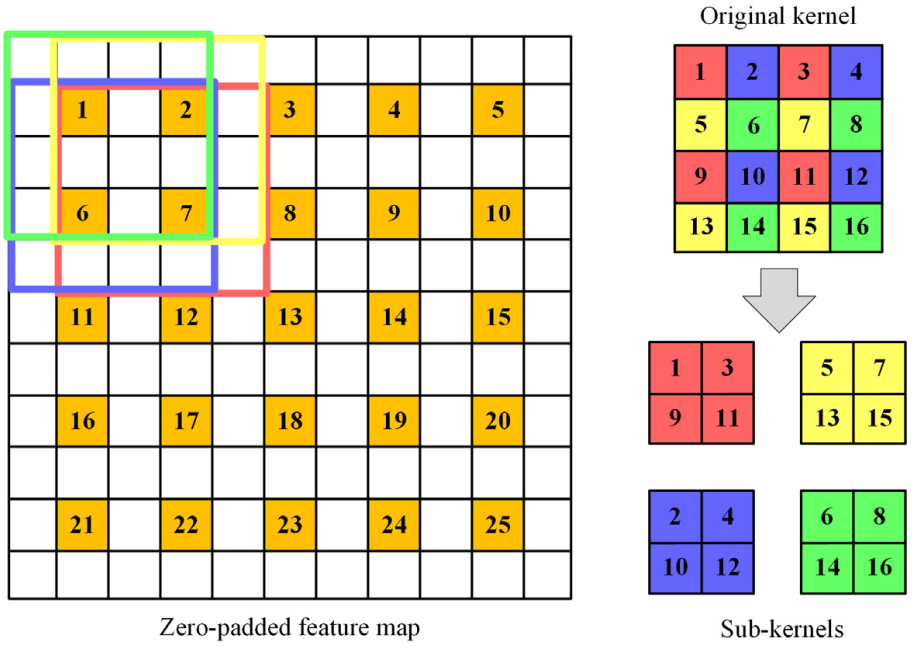

Figure 14. Sub-kernels for DeCONV.

After splitting the kernel, convolutions can be directly performed using the four sub-kernels, eliminating the zero-padding process and making DeCONV perfectly fit the dataflow of CONV. As shown in Figure 15, each new sub-kernel performs convolution with a stride of 1 on the original input feature map, and the results obtained by each sub-kernel are rearranged according to a fixed rule. Therefore, compared to CONV, this optimized DeCONV dataflow only requires additional logic for kernel-splitting and result rearrangement. This enables the universality of both the CONV and DeCONV dataflows in terms of the hardware resources and operating modes.

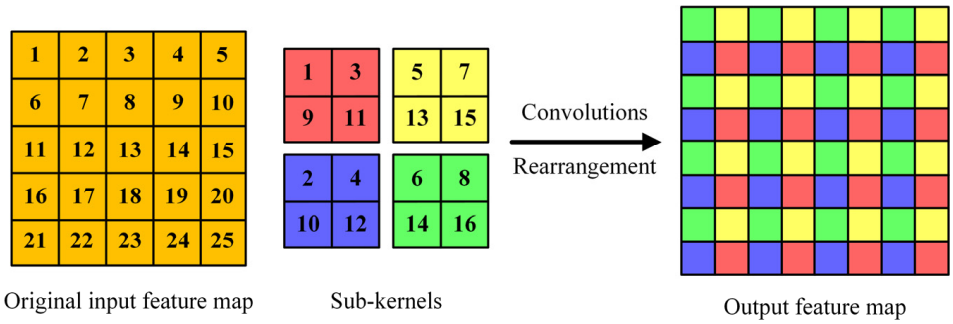

Figure 15. Computations of DeCONV without zero-padding.

The DeCONV method for kernels of other sizes follows a similar principle, but the difference lies in the sizes of the sub-kernels. For example, when the kernel size is 5×5, sub-kernels of different sizes are generated, as shown in Figure 16.

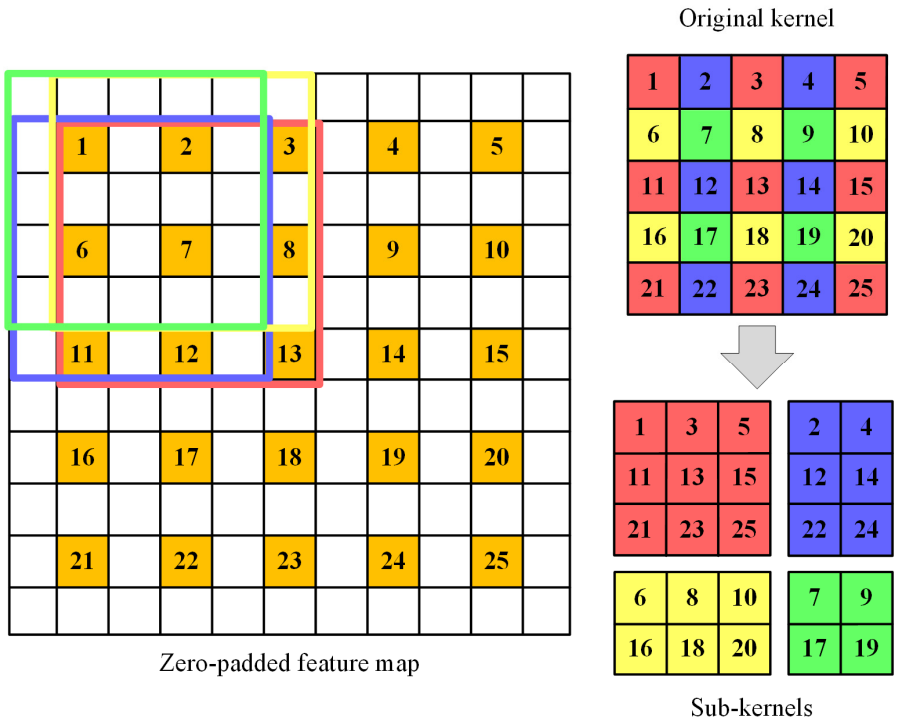

Figure 16. Generating different sizes of sub-kernels.

When new sizes of kernel are detected, reconfigurable control signals can be generated to selectively enable the registers and MAC logic in the PEs. Meanwhile, the sub-array can also shut down unnecessary PE rows via control signals. The rearrangement of the results follows the same pattern as described for the 4×4 kernel. Therefore, different types of DeCONV can be adapted to the proposed computing method above and mostly utilize the same hardware resources as CONV.

This design avoids unnecessary zero value computations and extra DeCONV module. It can perform highly parallel DeCONV with minimized dataflow and hardware heterogeneity. For different sizes of sub-kernels, the reconfigurable design also minimizes the power consumption and resource redundancy. In detail, the corresponding control signals are generated for different sizes of sub-kernels. Then, they dynamically configure the PEs and sub-arrays in the way shown in Figure 17.

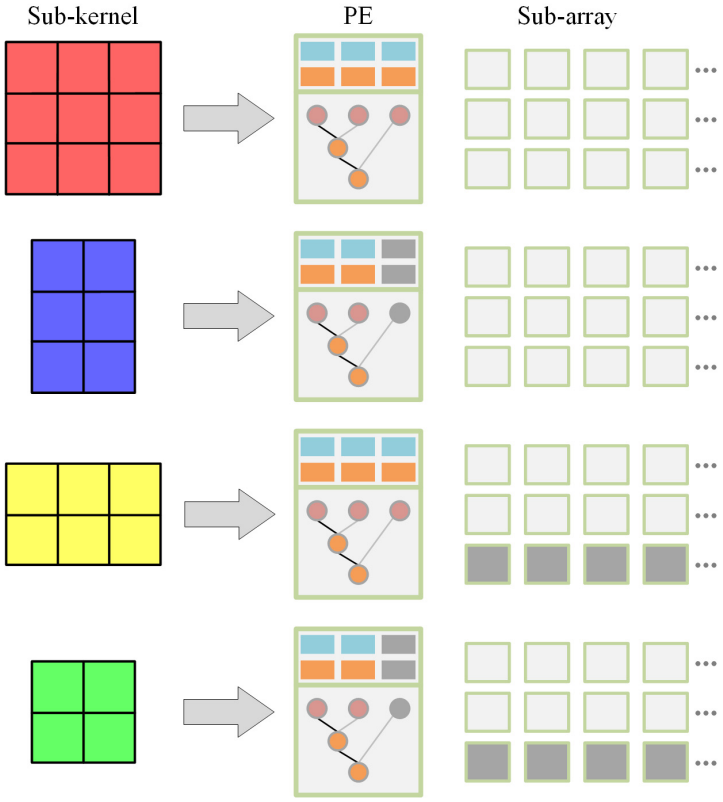

Figure 17. Reconfiguration of the PEs and sub-arrays when computing different sizes of sub-kernels.

Furthermore, the kernel-splitting can bring optimization benefits to hardware platforms that do not support large kernel sizes or have low performances. This is because the size of the kernel directly affects the number of MACs required in one sliding window. For the same performance, larger kernel sizes require more computational resources in the PE array.

4.3.2. The Designed Computing Strategy for the DeCONV

Four convolutions of four sub-kernels are performed during DeCONV. Therefore, each 3×14 sub-array can correspond to one sub-kernel. When the sub-kernel size is 2×2, the last row of each sub-array does not need to be enabled, and only 2 sets of registers and MAC units need to be enabled within the PE. When the sub-kernel has a different size, the enablement of the array can be dynamically adjusted accordingly. Although this mapping strategy sacrifices the input channel parallelism, for DeCONV, the convolution of the four sub-kernels is computed simultaneously, and the results can be combined and rearranged into one output map. This effectively utilizes the unique parallel dimension of DeCONV, further enhancing the sliding window parallelism. Additionally, computing the four sub-kernels simultaneously also facilitates data rearrangement. Therefore, in this accelerator design, DeCONV exhibits the same level of computational parallelism.

Since the results of the four sub-arrays correspond to different sub-kernels, the output buffer used during CONV cannot be used for accumulating the results between sub-arrays. Therefore, a special design is required. During DeCONV, four sets of independent FIFO buffers are used for accumulation. Each set contains 14 FIFOs with a depth of 128, which are responsible for accumulating the results between different channels of the four sub-kernels. Since each sub-array performs the accumulation of its own output, the amount of data that need to be accumulated in the FIFOs is relatively small. So, this additional buffer design for DeCONV computation occupies minimal BRAM resources. The proposed computing strategy for the DeCONV is shown in Figure 18.

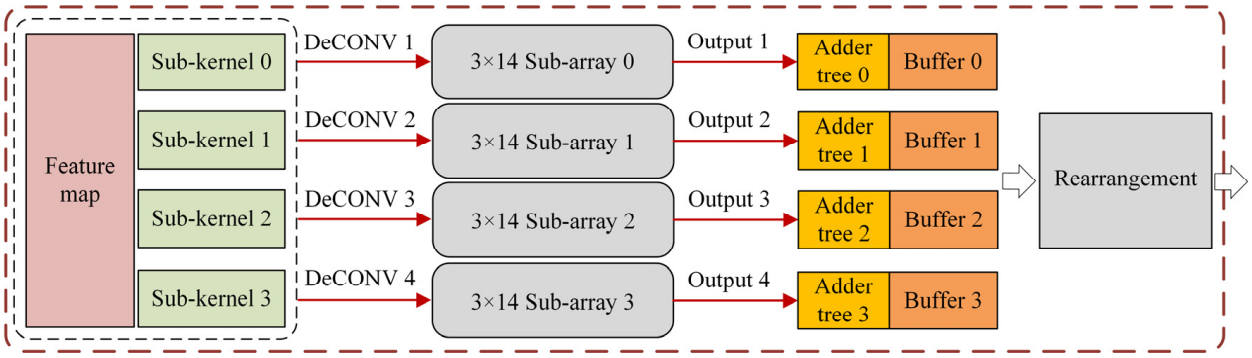

Figure 18. The on-chip computing strategy for the DeCONV.

A data rearrangement module is needed. This module rearranges the outputs from the four sub-arrays according to a certain pattern to form the complete output feature map. Due to the regularity of the data rearrangement, it only requires specific address allocation for the outputs. The rearranged data are then outputted and stored in the off-chip DRAM according to the generated addresses.

5. Experiment and Analysis

The proposed accelerator design primarily addresses the issues of incompatibility or low resource utilization and energy efficiency (performance per watt) exhibited by current mainstream designs when handling CONV, depthwise separable CONV, and DeCONV layers. To evaluate the optimization of this designed accelerator, the representative CNN models encompassing these three types of CONV layers were selected for experimental testing. The main focus was on resource utilization, performance (total latency), and power consumption during the execution of the CNN models. A comparative analysis was conducted against related state-of-the-art designs.

5.1. Implementation and Experimental Environment

For the experiment, CNN models were used to perform the deployment and operation of the proposed accelerator design. The designed hardware architecture was implemented using Xilinx Vivado 2022.1. After functional verification, synthesis and building constraints, the design was deployed on the Xilinx ZCU102 [26].

The trained and quantized parameter data of CNN models were loaded into the DDR4 memory included in the board. The PE array of this accelerator supports computation for a single CONV layer, and top control logics were designed for inter-layer data scheduling. The detail implementation on the FPGA experiment board is shown in Figure 19. The implemented accelerator can achieve a working frequency of 200 MHz and successfully perform CNN inference tasks. The utilization of basic resources in the FPGA platform for the proposed accelerator is shown in Table 1.

Table 1. Resources of the designed accelerator.

Resources	Usage (Our Design)	Usage (B1024/B4096)
LUT	32,130	33,796/51,351
DFF	35,214	48,144/98,818
DSP	896	230/710
BRAM	322	104/255

For CONV and depthwise separable CONV, this experiment conducted the deployment of two CNN models, VGG16 and MobileNetV1, as they mainly consist of CONV layers and depthwise separable CONV layers, respectively. The training of the required models was performed using the PyTorch framework and the ImageNet [27] dataset on an Nvidia RTX 2080Ti GPU platform. Weight data of the trained models were in FP32

format and were then quantized to obtain the INT8 data. The accuracy of the models was revalidated. The information of the used CNN models is shown in Table 2. As can be seen in the table, the parameters are effectively reduced with acceptable degradation of accuracy.

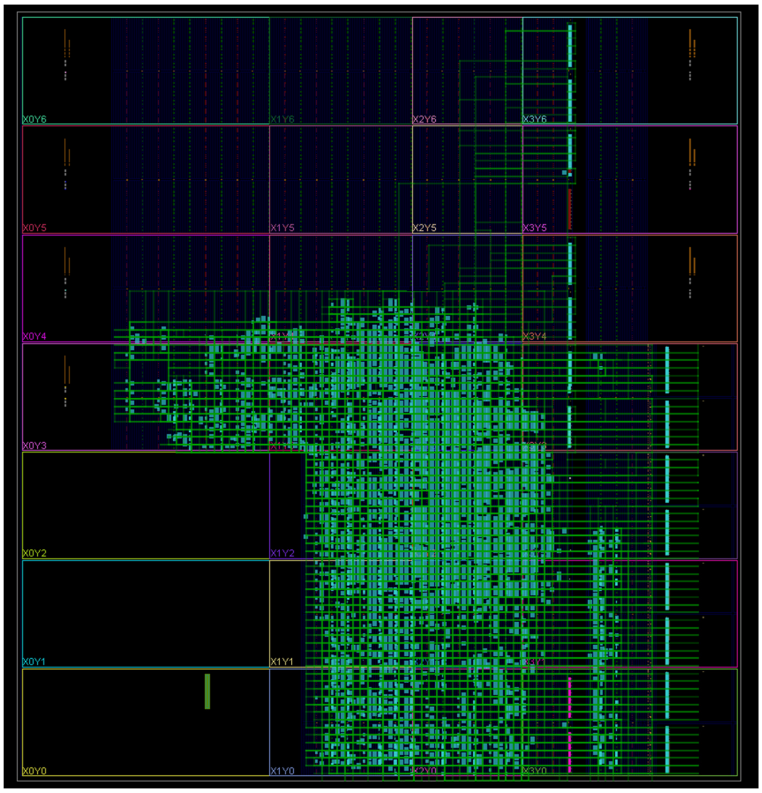

Figure 19. Hardware implementation on the ZCU102.

Table 2. Quantized CNN models for deployment.

Models	TOP-1/TOP-5 Accuracy (FP32)	Parameters (FP32)	TOP-1/TOP-5 Accuracy (INT8)	Parameters (INT8)
VGG16	0.704/0.887	56.12 MB	0.590/0.881	14.03 MB
MobileNetV1	0.702/0.879	12.16 MB	0.686/0.875	3.04 MB

5.2. Comparison and Results

The Xilinx DPU [23] was chosen, as a state-of-the-art generic CNN accelerator design, as a comparison. In the DPU acceleration platform, both depthwise separable CONV and DeCONV require additional hardware modules to be deployed on top of the acceleration architecture for standard CONV. For instance, in the B4096 DPU, handling depthwise separable CONV necessitates the addition of a depthwise CONV module composed of an extra 48 DSPs. The proposed architecture in this paper is able to perform depthwise separable CONV and DeCONV without the need for additional dedicated processing engines, thereby enhancing the compatibility of acceleration.

As shown in Figure 20, when running models primarily composed of depthwise separable CONV, the performance per watt of the DPU is noticeably lower compared to models dominated by standard CONV. Moreover, the performance disparity becomes larger with the increase in the DPU scale and parallelism. In the B4096 DPU, which has the highest resource usage and parallelism, the performance per watt of the YOLOv3 network is approximately 3.3 times that of the MobileNetV2. The performance per watt of depthwise separable CONV in DPUs of different parallelism consistently remains at a lower level, and

the resource utilization decreases as the usage of resources increases. This indicates that there is a resource utilization bottleneck for depthwise separable CONV in the DPU.

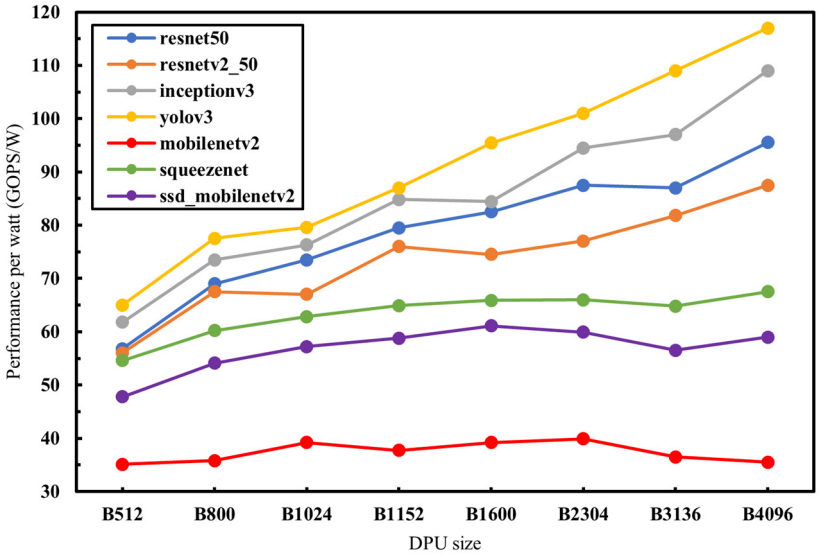

Figure 20. The performance per watt of different models in the DPUs.

The experiment focused on comparing the resource utilization and energy efficiency between standard CONV and depthwise separable CONV, emphasizing the efficiency variation of the system when switching between different models. The experiment measured the resource utilization and the performance per watt of the single-core B1024 DPU and B4096 DPU when running VGG16 and MobileNetV1, serving as a comparison with the proposed hardware architecture. Because that DPU uses the same INT8 quantization method, our design can achieve the same accuracy when deploying the same CNN models. The details of the experimental results are shown in Table 3.

Table 3. Comparative results when operating VGG16 and MobilenetV1.

Accelerators	Resource Utilization (VGG16/MobilenetV1)	Runtime (ms) (VGG16/MobilenetV1)	Performance per Watt (GOPS/W) (VGG16/MobilenetV1)
B1024	0.837/0.249	163.4/7.9	81.1/28.4
B4096	0.869/0.213	41.8/5.2	88.1/27.2
Ours	0.936/0.378	177.1/9.0	36.4/13.2

As the DPU is a commercial IP from Xilinx and incorporates unique enhancements such as the double data rate of the DSP and channel augmentation to improve performance, it outperforms our designed accelerator in terms of energy efficiency.

As shown in Table 3, when running VGG16, the resource utilization and energy efficiency of B1024 are lower than those of B4096. This is because the VGG16 is a CNN model with a high computational load. Thus, B4096 performs better with a higher degree of parallelism. When running MobilenetV1, the resource utilization and energy efficiency of B1024 are higher instead. This is because the computational load of light-weight CNN models is lower, a lot of resources of B4096 are redundant.

Our design has a smaller energy efficiency gap when running VGG16 and MobilenetV1 compared with the DPU. The parallelism and computational resources in the DPU are very large. Furthermore, depthwise CONV needs to be computed in a dedicated module, while pointwise CONV is computed in the main CONV engine. When running a standard CNN module such as VGG16, only the main CONV engine is used. But when running light-weight models, it is necessary to obtain the same high frequency in the dedicated module for depthwise CONV at the same time. This leads to additional power consumption

and a large gap when switching between these two scenarios. Instead, our design uses the same computation engine for depthwise CONV and standard CONV (pointwise CONV), which avoids the extra power consumption and improves the balance of energy efficiency.

From the comparative graph in Figure 21, it can be observed that our design achieved resource utilization optimizations of 1.08 times and 1.77 times when running VGG16 and MobileNetV1, respectively. When transitioning from VGG16 to MobileNetV1, the DPU experienced a 75% decrease in resource utilization and a 69% decrease in energy efficiency. In contrast, the proposed hardware architecture exhibited a lower decrease of 60% in resource utilization and a lower decrease of 64% in energy efficiency between these two network models. Therefore, combined with the advanced design and the low-power methods of Xilinx DPU, our architecture can further improve the efficiency of generic accelerators in complex scenarios.

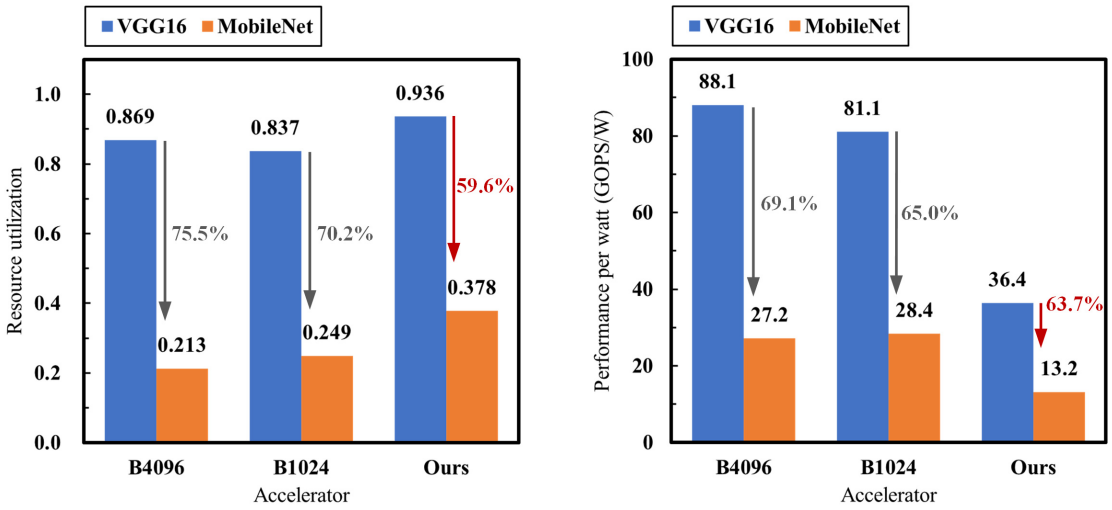

Figure 21. The comparative experiment when running VGG16 and MobileNetV1: (**a**) comparison of resource utilization; and (**b**) comparison of performance per watt.

To demonstrate the correct and efficient computation of the DCNN models without requiring additional hardware design, this paper selected three DeCONV layers with different parameters from Refinedet [28] for validation. The parameters of the DeCONV1, DeCONV2, and DeCONV3 layers in Refinedet are presented in Table 4. All three DeCONV layers upscale the original output image by a factor of two, and the output channels are all 256. For the proposed DeCONV dataflow in this paper, the total number of MACs is reduced to 25% of the original. The processing latencies for the three DeCONV layers are 0.031 ms, 0.11 ms, and 0.46 ms, respectively. The corresponding performance of the accelerator, which is evaluated by throughput, is 12.7 GOPS, 14.3 GOPS, and 13.7 GOPS. We also deploy these three DeCONV layers on B1024 as a comparison because of the similar resource usage. The comparative results are shown in Table 4. Although there is a gap in the runtime due to the difference of maximum frequency and design techniques, our work achieves higher resource utilization.

Table 4. Comparative results when operating DeCONV layers.

Layers	Input Size	Output Size	Kernel Size	Stride	Runtime (ms) (Ours/B1024)	Resource Utilization (Ours/B1024)
DeCONV1	$6 \times 8 \times 4$	$12 \times 16 \times 256$	2	2	0.031/0.017	0.821/0.783
DeCONV2	$12 \times 16 \times 4$	$24 \times 32 \times 256$	2	2	0.11/0.08	0.867/0.809
DeCONV3	$24 \times 32 \times 4$	$48 \times 64 \times 256$	2	2	0.46/0.31	0.872/0.820

6. Conclusions

Based on the existing complex scenarios and limited compatible accelerators, this study designs a reconfigurable generic accelerator architecture, a tiling computational dataflow, and a unified computing scheme on the basis of conventional acceleration designs. This design allows dynamic reconfiguration when standard CONV, depthwise separable CONV, and DeCONV layers are operated, minimizing the resource and power redundancy of hardware platforms. Compared to the state-of-the-art DPU B4096 accelerator, this study achieves higher resource utilization and lower utilization and energy efficiency gap between a standard CNN and lightweight CNN. For tasks involving multiple CNN models in complex scenarios, this design ensures a higher balance, allowing better utilization of the hardware resources in edge devices and lower power fluctuations while maintaining certain performance. In complex tasks such as multimodal sensor fusion, low-light image enhancement, object recognition, and tracking, where CNN models need to be deployed alternately, the designed partial reconfiguration minimizes the fluctuations in the hardware resource allocation and power, avoiding resource redundancy and performance bottlenecks.

By combining advanced on-chip design, the performance per watt can be further improved, ensuring a high balance while achieving excellent acceleration efficiency and performance. Moreover, complex scenarios not only involve alternating between different types of CNN models but also switching between CNN models with different computational and storage loads. Based on this design, a set of configurations with different computational parallelism can be introduced. Referring to the configuration division of the DPU, multiple configurations with different parallelism and performance can be designed. Different types of accelerators can be configured in the scalability of the number of MACs, size of the PE array and number of sub-arrays. So, there is a set of different accelerators. Meanwhile, since the DPU cannot multi-core match different configurations, designs can optimize the dataflow allocating logics to make it possible to match different PE computing arrays together. For more complex and larger scenarios, combinations of different configurations in the accelerator set provide a very high degree of flexibility and the most efficient hardware deployment. The use of Xilinx's dynamic function exchange (dfx) [29] technology allows more flexible and rapid hardware dynamic reconfiguration. Combined with software prediction [30] for CNN task loads and optimal configuration selection, the balance and flexibility of the accelerator in complex CNN applications can be further enhanced, achieving greater resource utilization and performance per watt on edge platforms.

Author Contributions: Conceptualization, H.T., K.H., S.H. and Y.L.; methodology, H.T., S.H. and Y.L.; software, H.T.; validation, H.T.; formal analysis, H.T.; investigation, H.T., K.H., S.H. and Y.L.; resources, K.H. and S.H.; data curation, H.T. and Y.L.; writing—original draft preparation, H.T.; writing—review and editing, H.T. and S.H.; visualization, H.T.; supervision, K.H., S.H. and Y.L.; project administration, K.H. and S.H.; funding acquisition, K.H. and S.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Fundamental Research Funds for the Central Universities grant number 10823544.

Data Availability Statement: Data are contained within this article.

Acknowledgments: This research was supported by the School of Electronic Engineering, Beijing University of Posts and Telecommunications and Fundamental Research Funds for the Central Universities.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this article:

CNN	Convolutional Neural Network
ADAS	Advanced Driver Assistance System
CONV	Convolution
DCNN	Deconvolutional Neural Network
DeCONV	Deconvolution
FPS	Frame Per Second
IOM	Input-Oriented Method
TDC	Transforming Deconvolution into Convolution
GOPS	Giga Operations Per Second
FPGA	Field Programmable Gate Array
DPU	Deep-Learning Processing Unit
WS	Weight Stationary
IS	Input Stationary
FLOP	Floating Point Operations
PE	Processing Element
MAC	Multiply–Accumulate
Psum	Partial Sum
IP	Intellectual Property
BRAM	Block Random Access Memory
DRAM	Dynamic Random Access Memory
DDR	Double Data Rate
LUT	Lookup Table
DFF	D-Flip-Flop
DSP	Digital Signal Processor
FP32	32-bit Floating Point
INT8	8-bit Fixed Point

References

1. Badue, C.; Guidolini, R.; Carneiro, R.V.; Azevedo, P.; Cardoso, V.B.; Forechi, A.; Jesus, L.; Berriel, R.; Paixao, T.M.; Mutz, F. Self-Driving Cars: A Survey. *Expert Syst. Appl.* **2021**, *165*, 113816. [CrossRef]
2. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
3. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. Mobilenetv2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
4. Feizi, S.; Marbach, D.; Médard, M.; Kellis, M. Network Deconvolution as a General Method to Distinguish Direct Dependencies in Networks. *Nat. Biotechnol.* **2013**, *31*, 726–733. [CrossRef] [PubMed]
5. Noh, H.; Hong, S.; Han, B. Learning Deconvolution Network for Semantic Segmentation. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1520–1528.
6. Long, J.; Shelhamer, E.; Darrell, T. Fully Convolutional Networks for Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3431–3440.
7. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
8. Feng, D.; Haase-Schütz, C.; Rosenbaum, L.; Hertlein, H.; Glaeser, C.; Timm, F.; Wiesbeck, W.; Dietmayer, K. Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 1341–1360. [CrossRef]
9. Li, C.; Guo, C.; Han, L.; Jiang, J.; Cheng, M.-M.; Gu, J.; Loy, C.C. Low-Light Image and Video Enhancement Using Deep Learning: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 9396–9416. [CrossRef]
10. Chen, Y.-H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J. Solid-State Circuits* **2016**, *52*, 127–138. [CrossRef]
11. Chen, Y.-H.; Emer, J.; Sze, V. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 367–379. [CrossRef]
12. Chen, Y.-H.; Yang, T.-J.; Emer, J.; Sze, V. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2019**, *9*, 292–308. [CrossRef]

13. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; pp. 1–12.
14. Su, J.; Faraone, J.; Liu, J.; Zhao, Y.; Thomas, D.B.; Leong, P.H.; Cheung, P.Y. Redundancy-Reduced Mobilenet Acceleration on Reconfigurable Logic for Imagenet Classification. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 14th International Symposium, ARC 2018, Santorini, Greece, 2–4 May 2018, Proceedings 14*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 16–28.
15. Wu, D.; Zhang, Y.; Jia, X.; Tian, L.; Li, T.; Sui, L.; Xie, D.; Shan, Y. A High-Performance CNN Processor Based on FPGA for MobileNets. In Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, 8–12 September 2019; pp. 136–143.
16. Zhao, R.; Niu, X.; Luk, W. Automatic Optimising CNN with Depthwise Separable Convolution on FPGA: (Abstact Only). In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; p. 285.
17. Xie, X.; Sun, F.; Lin, J.; Wang, Z. Fast-ABC: A Fast Architecture for Bottleneck-like Based Convolutional Neural Networks. In Proceedings of the 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA, 15–17 July 2019; pp. 1–6.
18. Bai, L.; Zhao, Y.; Huang, X. A CNN Accelerator on FPGA Using Depthwise Separable Convolution. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 1415–1419. [CrossRef]
19. Yu, Y.; Zhao, T.; Wang, K.; He, L. Light-OPU: An FPGA-Based Overlay Processor for Lightweight Convolutional Neural Networks. In Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, 23–25 February 2020; pp. 122–132.
20. Zhang, X. *A Design Methodology for Efficient Implementation of Deconvolutional Neural Networks on an FPGA*; University of California: San Diego, CA, USA, 2017.
21. Yan, J.; Yin, S.; Tu, F.; Liu, L.; Wei, S. GNA: Reconfigurable and Efficient Architecture for Generative Network Acceleration. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 2519–2529. [CrossRef]
22. Chang, J.-W.; Kang, K.-W.; Kang, S.-J. An Energy-Efficient FPGA-Based Deconvolutional Neural Networks Accelerator for Single Image Super-Resolution. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, *30*, 281–295. [CrossRef]
23. Xilinx DPUCZDX8G for Zynq UltraScale+MPSoCs Product Guide (PG338). Available online: https://docs.xilinx.com/r/en-US/pg338-dpu (accessed on 5 January 2024).
24. Ezilarasan, M.R.; Britto Pari, J.; Leung, M.-F. Reconfigurable Architecture for Noise Cancellation in Acoustic Environment Using Single Multiply Accumulate Adaline Filter. *Electronics* **2023**, *12*, 810. [CrossRef]
25. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.
26. Xilinx ZCU102 Evaluation Board User Guide (UG1182). Available online: https://docs.xilinx.com/v/u/en-US/ug1182-zcu102-eval-bd (accessed on 5 January 2024).
27. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M. Imagenet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252. [CrossRef]
28. Zhang, S.; Wen, L.; Bian, X.; Lei, Z.; Li, S.Z. Single-Shot Refinement Neural Network for Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4203–4212.
29. Vivado Design Suite User Guide: Dynamic Function eXchange (UG909). Available online: https://docs.xilinx.com/r/zh-CN/ug909-vivado-partial-reconfiguration (accessed on 5 January 2024).
30. Han, K.; Luo, Y. Feasibility Analysis and Implementation of Adaptive Dynamic Reconfiguration of CNN Accelerators. *Electronics* **2022**, *11*, 3805. [CrossRef]