Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 4726652, 23 pages
https://doi.org/10.1155/2022/4726652

Research Article

Deep-Sea: A Reconfigurable Accelerator for Classic CNN

Hao Xiong,[1,2] **Kelin Sun**,[1] **Bing Zhang**,[1] **Jingchuan Yang**,[1] **and Huiping Xu**[1]

[1]*Institute of Deep-Sea Science and Engineering, Chinese Academy of Sciences, Hainan 572000, China*
[2]*School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Kelin Sun; sunkl@idsse.ac.cn

Received 9 September 2021; Revised 10 November 2021; Accepted 9 December 2021; Published 2 February 2022

Academic Editor: Danfeng Hong

Copyright © 2022 Hao Xiong et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

To meet the changing real-time edge engineering application requirements of CNN, aiming at the lack of universality and flexibility of CNN hardware acceleration architecture based on ARM+FPGA, a general low-power all pipelined CNN hardware acceleration architecture is proposed to cope with the continuously updated CNN algorithm and accelerate in hardware platforms with different resource constraints. In the framework of the general hardware architecture, a basic instruction set belonging to the architecture is proposed, which can be used to calculate and configure different versions of CNN algorithms. Based on the instruction set, the configurable computing subsystem, memory management subsystem, on-chip cache subsystem, and instruction execution subsystem are designed and implemented. In addition, in the processing of convolution results, the on-chip storage unit is used to preprocess the convolution results, to speed up the activation and pooling calculation process in parallel. Finally, the accelerator is modeled at the RTL level and deployed on the XC7Z100 heterogeneous device. The lightweight networks YOLOv2-tiny and YOLOv3-tiny commonly used in engineering applications are verified on the accelerator. The results show that the peak performance of the accelerator reaches 198.37 GOP/s, the clock frequency reaches 210 MHz, and the power consumption is 4.52 w under 16-bit width.

1. Introduction

With the development of deep convolutional neural networks, various new convolutional neural network algorithm structures emerge endlessly. Compared with traditional image processing and analysis algorithms, such as SIFT, SVM, and KNN, the key parameters of DCNN-based processing algorithms are obtained based on specific data set training, which shows better robustness and algorithm accuracy. Therefore, DCNN-based processing algorithms are now widely used in various directions and related fields of image processing and analysis, such as target detection and recognition, video surveillance, automatic driving, assisted driving, and damage detection. In some areas where the development is lagging, such as in the more special deep-sea exploration field, due to the lag in the development of image processing and analysis technologies and computing platforms for landers, AUVs, ROVs, and live video landers in deep-sea scenes, at present, most signal processing is still based on manual control and centralized calculation of surface mother ship, and acoustic information is mainly used as the main information source of environmental perception. The existence of these problems, on the one hand, makes the optical image information unable to directly serve the environment perception of underwater equipment, which limits the intelligent process of underwater equipment; on the other hand, the transmission bandwidth is occupied by a large amount of invalid data, which limits the three-dimensional deployment of the detection system in a large area and a wide range. Nowadays, the rapid development of the DCNN algorithm and edge computing can bring new ideas to solve these problems, so that optical image information can better serve the intelligent and large-scale deployment of deep-sea equipment. However, the continuous update of the DCNN network structure promotes the continuous improvement of the algorithm performance, but this is at the cost of a substantial increase in the network calculation scale, resulting in many algorithms not entering the actual engineering application stage. Therefore, research on network lightweighting is also developing rapidly, such

as MobileNet, YOLO-tiny series, and other algorithms. Compared with ordinary DCNN algorithms, some lightweight DCNN algorithms have a greatly reduced amount of calculation, and the accuracy can also meet the requirements of engineering applications.

However, compared with traditional algorithms, these lightweight algorithms are still computationally consuming algorithms, and at the same time, they are in the stage of rapid iteration. Therefore, studying how to provide sufficient computing power for a suitable network to meet the computing requirements of actual edge engineering tasks and the rapid adaptation requirements of different algorithm structures has become the key to whether deep convolutional neural network algorithms can be applied to specific engineering applications. At this stage, the mainstream deep convolutional neural network computing platforms mainly include CPU, GPU, ASIC, and FPGA.

As the first processor to deploy deep learning, the CPU has strong generality, but it is executed sequentially. When faced with parallel deep convolutional neural networks among channels, CPU computing efficiency is not high. With the increase in the scale of convolutional neural networks, whether it is an embedded CPU or a desktop CPU, their computing power is not enough to support the needs of algorithm engineering applications, and they cannot provide efficient data operations and memory access requirements for computationally intensive tasks. Therefore, both the academic circle and the industry have turned their attention to high-performance computing platforms such as GPU and FPGA. GPU is good at floating-point arithmetic and parallel arithmetic due to its high parallel computing unit and large memory bandwidth [1, 2]. The convolutional neural network's various computing channels are parallel, so it is very suitable for using GPU for computing. Coupled with its supporting parallel computing software architecture, such as CUDA, it can greatly reduce the algorithm development cycle. However, GPU's high energy consumption and large area characteristics make it unable to undertake edge computing tasks [3, 4]. Even for embedded GPUs, its uncertain delay and large power consumption will bring difficulties to practical applications and cannot provide high-speed serial communication, real-time processing, and low power consumption at the same time [5, 6]. Therefore, neither CPU nor GPU can meet the needs of scenarios such as underwater unmanned submersibles, underwater live broadcast systems, buoys, AUVs, and ROVs. Besides, the design and verification cycle of ASIC is too long, the capital investment is too large, and the flexibility is insufficient. In contrast, FPGA has the advantages of fixed delay, low power consumption, high flexibility, low R&D cost, real-time processing, and low-latency communication, making it one of the best solutions for real-time edge computing scenarios [7]. However, FPGA bandwidth and on-chip storage resources are limited, and large-scale convolutional neural networks cannot be fully deployed on it. As a result, advanced algorithms based on convolutional neural networks cannot be directly deployed on real-time embedded vision systems. Therefore, to make up for the limited memory and computing resources, more and more researchers are researching FPGA-side convolutional neural network acceleration architecture. However, it is very difficult to design complex calculations using FPGAs, which are more suitable for parallel computing scheduling. Therefore, there are more and more researches on heterogeneous acceleration schemes, using RISC CPU for system access control and FPGAs for convolution neural networks accelerate research.

The main challenges of deploying DCNN on FPGA are parallel optimization of convolutional computation, optimization of on-chip and off-chip memory access, and optimization of generality. The former two are the main factors that affect the forward inference time of DCNN, and the latter is mainly because the algorithm deployment on FPGA is too difficult to reduce the cycle of accelerator adaptation to different algorithms and accelerate application landing. To solve these problems, many researchers have made efforts in three aspects: algorithm pruning, quantification, and hardware architecture design. [8–11] mainly start from model quantization and pruning, reduce the parameter bit width and the number of parameters, to achieve the purpose of reducing the amount of calculation, making the algorithm more suitable for hardware implementation. [12, 13] designed special accelerators for convolutional neural networks from the perspective of computational parallelism, computational unit structure, and memory access optimization. These accelerators have one of the following characteristics: insufficient circuit optimization of the accelerator leads to low calculation frequency; insufficient generality of the accelerator can only provide adaptation for a fixed algorithm; the input size of the accelerator and other related calculation parameters are fixed, and the flexibility is not high; frequent access to off-chip memory or consumption of a large amount of on-chip storage increases power consumption. Based on HLS, [14] designed a dedicated accelerator for the YOLO series of algorithms, which has a certain improvement in generality and development cycle. However, due to the insufficient optimization of the circuit by HLS, its computing performance cannot meet the needs of low-latency engineering applications. Therefore, for FPGA, it is necessary to study the dynamic balance among algorithm inference performance, power consumption, and flexibility according to different application requirements [15]. To alleviate the above problems, this paper proposes a reconfigurable and versatile DCNN acceleration architecture, which not only satisfies high computing performance but also improves the configurability and scalability of the accelerator. We perform Verilog modeling and critical path optimization based on the AXI protocol standard. The accelerator is currently able to adapt to the computing requirements of the mainstream DCNN algorithm and at the same time can achieve a better energy efficiency ratio and computing efficiency. The main contributions of this article are as follows:

(1) According to the structural characteristics of the DCNN algorithm, a reconfigurable and scalable general CNN acceleration workflow is proposed, and the accelerator is modeled at the RTL level. This workflow can easily perform model extraction and

memory planning for the target DCNN algorithm and map it to the target device for accelerated calculation. Its characteristic is that it can dynamically balance the relationship among the DCNN algorithm scale, on-chip computing and storage resources, computing performance, and power consumption according to specific scene constraints. Compared with the existing DCNN accelerated workflow, the workflow proposed in this article has been further optimized and improved in four aspects: computing core, control instructions, on-chip cache architecture, and off-chip memory access management. The generality and processing bandwidth of accelerators reduce the power consumption of accelerators, which is embodied in the improvement of computing density and energy efficiency ratio

(2) In terms of computing core, this paper divides the classic convolutional neural network algorithm into the hardware design, and according to the method of loop unrolling, combined with the instruction set design, the scheduling rules are designed for the layers of the convolution neural network and among the layers. Under the scheduling rules, the calculation requirements of a single convolutional layer can be completed by repeatedly scheduling the convolution core, and then, the calculation of the entire network is completed, achieving the purpose of hardware parallel acceleration of the convolutional neural network at the edge

(3) A set of CNN basic computing instruction set is proposed, which is used for computing flow control, parallel computing control, filling control, pooling control, data and parameter reading and writing control, memory mapping control, etc., of the general acceleration architecture. Compared with the existing reconfigurable computing architecture, the instruction set designed in this article supplements the fine-grained instruction set and static instructions, which solve the dynamic computing configuration and hardware resource allocation of large-scale algorithms on resource-constrained platforms

(4) A set of on-chip caching strategies is proposed, which uses on-chip caching to separate the calculation of convolution and postconvolution processing and adds ping-pong operation. Compared with the existing accelerator computing core, the on-chip buffering method proposed in this paper not only solves the problem of on-chip storage of intermediate convolution results but also preprocesses the convolution results to achieve parallel data stream acquisition in the postconvolution processing part. Such processing can not only increase the processing bandwidth of the computing core but also provide a high-speed parallel data stream for the postconvolution processing calculation, which improves the overall calculation efficiency

(5) At this stage, the existing accelerator system will basically choose Xilinx official DMA as the memory access controller, but the resource occupation of DMA is redundant, and register configuration is required before using, and it is not compatible with the accelerator's unique memory data arrangement mode. Therefore, this paper proposes a memory management system and adapts it to the corresponding PC-side python program. The system can flexibly change the arrangement of data in the memory and flexibly access data and parameters, providing a data drive for the overall performance of the entire acceleration system

This article mainly consists of the following sections. The second section mainly reviews the work related to deep convolutional neural network algorithms and hardware acceleration. The third section mainly introduces the basic theoretical basis of deep convolutional neural network acceleration. The fourth section mainly introduces the overall architecture of the accelerator and the design of each component module. The fifth section mainly analyzes and compares the experiments of the acceleration system. The sixth part summarizes the work of this paper and introduces future work.

2. Related Work

The research on deep convolutional neural networks mainly focuses on the research of data processing and analysis algorithms based on DCNN and the research of computational acceleration. Regarding algorithm research, it is mainly divided into two parts, one is the research on the accuracy of algorithm processing, and the other is the research on the lightweight of the algorithm. Regarding the research of computing acceleration, it mainly focuses on the research of computing architecture, algorithm quantification, algorithm pruning, etc. In other words, the research on hardware acceleration is about how to make the computing architecture more universal and to further explore the computing power on the limited hardware resources. Below, let us review the representative work on algorithm model and calculation acceleration.

2.1. Algorithm Model. With the substantial increase in hardware computing power and the development of big data, in 2012, British computer scientists Hinton, Alex Krizhevsky released AlexNet et al. with a network depth of 7. Its appearance caused a sensation in academia and industry. Subsequently, deep convolutional neural networks began to develop rapidly. Researchers in various research fields have applied deep convolutional neural networks to their research in this field, and a large number of traditional algorithms have been replaced.

In the field of traditional image processing and analysis, deep convolutional neural networks have achieved disruptive effects in most directions. Different from the traditional manual design of feature extraction templates, deep convolutional neural networks can learn features based on data

sets, thereby automatically extracting features and proceeding to the next step. A lot of advanced work on image classification, image segmentation, target detection, etc. has emerged at this stage. In [16], the authors innovatively proposed the YOLO target detection method, which converts the target detection process into a regression problem, so that the detection algorithm can be optimized end-to-end. Compared with the two-stage algorithm, the processing speed of this type of one-stage algorithm is very obvious, and the accuracy of the algorithm can meet the requirements of practical applications. Reference [17] is aimed at the problem of semantic segmentation with limited cross-channel data in large-scale urban scenes and introduces self-generating confrontation network and mutual generation confrontation network modules into the existing multimodal deep neural network, to obtain more effective and robust information transmission. About the disadvantages of single-channel classification tasks, [18] innovatively proposed a general multimodal deep learning framework (MDL), which provides a solution for complex scenes that require fine classification. In [19], because of the large amount of calculation when the traditional graph convolutional neural network is applied to large-scale remote sensing problems, a method for training large-scale graph neural networks in small batches is proposed. After a lot of experiments, it is proved that this method is better than the traditional graph convolutional neural network.

In more detailed underwater image processing and analysis, due to the limitations of traditional methods, the problem of underwater image degradation has not been properly resolved, which is the main factor that affects the subsequent analysis results of the image. Therefore, some researchers apply deep convolutional neural networks to underwater image preprocessing. Experiments have proved that this implicit method does bring a relatively better enhancement effect. For example, a research team from the Institute of Deep-Sea Science and Engineering of the Chinese Academy of Sciences once developed a network model for deep-sea image enhancement based on the U-Net backbone network structure, which eased the problem of deep-sea image enhancement to a certain extent. This model not only enhances the visual effect of underwater images but also improves the performance of the subsequent image analysis process. Another example is literature [20], which changed the traditional image enhancement and detection strategy as two independent modules without interaction and innovatively proposed a fusion model of underwater image enhancement and detection. Its performance exceeds network models such as UDCP and DUIENet.

In summary, deep convolutional neural networks have almost ruled most of the directions of computer vision. However, in the actual engineering system, the requirements for power consumption, area, and performance often need to be compromised. Therefore, it is necessary to study the calculation acceleration model that makes the above three indicators more balanced. The following is a general introduction to some representative work of current computing acceleration.

2.2. DCNN Acceleration. With the continuous improvement of the performance of deep convolutional neural networks, the structure of the network model is gradually bloated. The amount of calculation is growing rapidly, and the amount of parameters is becoming more and more huge. Although the performance of the network model has been significantly improved under the increase of these parameters, the problem of inefficiency follows, that is, the number of parameters of the network model and the speed of the forward inference process of the network model. In response to this problem, many scholars have done a lot of research on DCNN computing acceleration.

Ma et al. developed a scalable and extensible training framework, which can use GPU to accelerate the training of deep learning models based on data parallelism at the nodes in the cluster [21]. Both synchronous and asynchronous training are implemented in the framework, and the parameter exchange among GPUs is based on CUDA-aware MPI. In [22–25], the team led by Chen Yunji of the Chinese Academy of Sciences launched a series of ASIC-based convolutional neural network accelerators. The energy efficiency of the PuDianNao chip in this series is 60 times that of the previous most advanced convolutional neural network accelerator. The power consumption under the 65 nm process is only 320 mW, and the area is 4.86 mm^2, but it is still about 30 times faster than the high-end GPU. Meloni et al. proposed a programmable accelerator suitable for medium- and high-grade FPGA. By programming some memory mapping controllers, the convolution operation with different convolution core sizes and steps can be adapted to different convolutions through software configuration at runtime. It has high execution efficiency when dealing with $3 * 3$ small convolution kernels [26]. Zhang et al. proposed a deep-flowing multi-FPGA computing architecture. This solution maps different layers to different FPGA development boards through the characteristics of neural network layer-to-layer computing serialization to solve the problem of insufficient resources on a single FPGA. Through multiple low-end FPGAs connected in series, the performance is similar to that of high-end GPUs, and the power consumption is lower than that of GPUs [27].

This paper focuses on creating a reconfigurable deep convolutional neural network accelerator under the edge engineering application scenario, using the deterministic delay and reconfigurability of FPGA, and named it deep-sea.

3. CNN Acceleration Principle

3.1. CNN. The application of CNN in the field of computer vision is very successful at this stage. It is an important branch of deep learning and belongs to a supervised learning algorithm. For the general convolutional neural network, its application mainly includes two stages: algorithm training stage and algorithm inference stage. The former is mainly the process of network learning features, which is different from the traditional method of manually designing feature extraction templates, and realizes the autonomous learning of features. The latter is the key to the application, for a certain task, and when the training phase is over, the weight

parameters of its feature extraction will be fixed, and the remaining work is to input the data of the task to be executed into the network for inference. The focus of this paper is to optimize the task calculation density, energy efficiency, and deployment efficiency of inference in edge devices. As shown in Figure 1, it mainly shows the main stages of inference of deep convolutional neural networks, from the input of image data to the convolution with weight data, and then the activation function and pooling. When calculations of all convolutional layers are completed, classification is performed to obtain the final result. In short, it is the two stages of feature extraction and classification.

3.2. Related Methods. In recent years, with the continuous improvement of DCNN performance, the algorithm structure is constantly changing, and the number of network layers and the scale of parameters are constantly increasing. From the original 8-layer, 250 M AlexNet, to the later 16-layer, 500 M VGG, and then to the following hundreds of layers of network scale, the large increase in the number of calculations and parameters of the network poses new challenges to the versatility of the computing architecture, as well as new challenges to the computing performance and storage bandwidth of edge computing. Therefore, it is necessary to consider the versatility, parallelism, and bandwidth adaptation of the hardware computing architecture.

The main calculation process of DCNN includes convolution, pooling, activation, and normalization. Suppose that m is the number of batches, n is the number of output channels, r and v are the size parameters of the feature map, C is the depth of the input feature map, and J and L are the size of the convolution kernel, respectively. The following will analyze each calculation process, as shown in Equation (1), which expresses the convolution calculation process of feature map and weight. From the expression, it can be seen that the three-dimensional summation process can be processed in parallel, plus the convolution process is performed in parallel, so it can be processed in parallel from at least four dimensions. This also constitutes the most basic parallel acceleration principle of the DCNN accelerator. It is also worth noting that the convolution process of each pixel is closely related to the number of input and output channels, pixel location, and feature map size. Therefore, these parameters are a key part of constructing a general architecture instruction set.

$$Y[m, n, r, v] = B[n] + \sum_{c=1}^{C} \sum_{j=1}^{J} \sum_{l=1}^{L} X[m, c, r + j, v + l] \cdot K[n, c, j, l]. \tag{1}$$

Equations (2)–(4) are, respectively, activation function, pooling operation, and batch normalization operation. It can be seen from its expression that the activation operation and batch normalization operation are direct operations on the convolution results of a single output channel, so fine-grained pipeline operations can be performed after the convolution results are completed to achieve the purpose of high-performance computing.

$$R[m, n, r, v] = \text{Relu}(X[m, n, r, v]), \tag{2}$$

$$Y[m, n, r, v] = \frac{X[m, n, r, v] - \mu}{\sqrt{\sigma^2 + \varepsilon}} \gamma + \beta. \tag{3}$$

Equation (4) is the expression of the pooling operation, and its area of action is within the scope of the K domain. The hardware architecture design of most systems for this operation is still a pipeline method. This method is obviously for discontinuous data flow, which is not the best choice. Therefore, in the subsequent architecture design, this paper uses on-chip storage resources to preprocess the feature map to achieve the purpose of accelerating the parallel processing of the four-domain convolution results.

$$P[m, n, r, v] = \underset{t,i \in [1:K]}{\text{Max}} (X[m, n, r + t, v + i]). \tag{4}$$

The above is an analysis of accelerated parallelism, and to achieve the purpose of a general architecture, it is also necessary to extract the union of the instruction set from the above algorithm expressions to guide the calculation. In addition, in terms of bandwidth optimization, this paper uses convolution kernel multiplexing, feature map multiplexing, window multiplexing, and on-chip caching strategies to greatly increase the data multiplexing rate on-chip and ease the bandwidth pressure of off-chip storage. These specific design ideas will be embodied in the following discussion.

4. Accelerator Design

Affected by the huge amount of calculation of the convolutional neural network, the on-chip computing and storage resources of the FPGA have become the main limiting factors for the reconfigurable accelerator. The best solution is to reuse limited on-chip resources and replace space with time to complete the convolutional neural network calculation. In this case, the versatility and efficiency of the system architecture are particularly important, so this part will focus on the design thinking applied in the deep-sea design phase.

4.1. System Architecture. The system architecture has a very large impact on the overall performance of the system, and the main factors that affect a system architecture are the design of the computing architecture, memory access architecture, and instruction architecture. The system architecture proposed in this paper is a heterogeneous computing architecture. A fully pipelined, partially parallel reconfigurable computing architecture is composed of CPU and FPGA. Figure 2 shows the block diagram of the heterogeneous system architecture. On the whole, the architecture is mainly divided into a CPU part and an FPGA part. The CPU is the overall control unit of the entire system architecture and is mainly responsible for calculations or operations that require little computing performance and are difficult to implement in hardware. FPGA is mainly responsible for the computational consumption part of the entire system architecture.

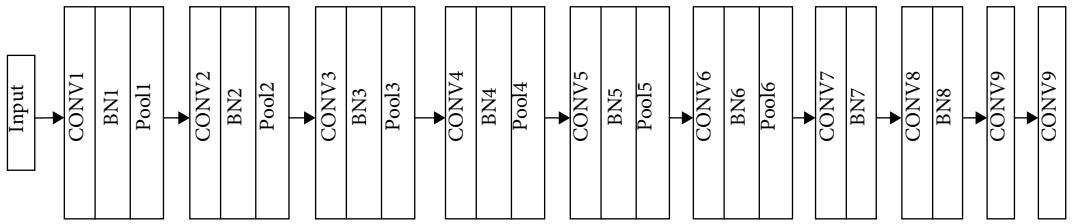

Figure 1: The structure diagram of YOLOv2-tiny algorithm.

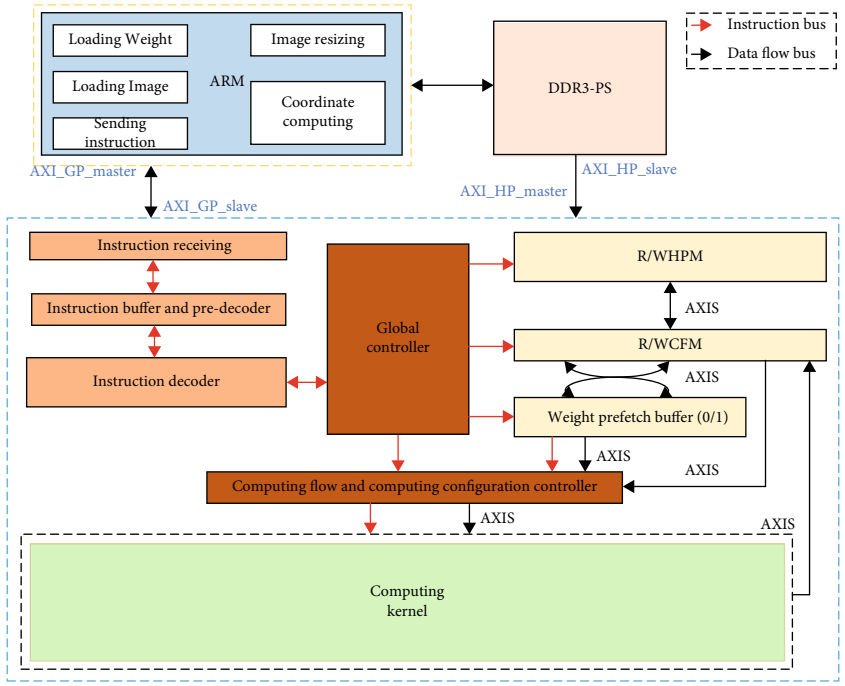

Figure 2: System architecture.

The above discrete subsystems are configured by their respective local controllers for work-mode configuration and complete specific work tasks in the form of pipeline under the scheduling of the global controller. The workflow of the entire system architecture is as follows:

(1) Extract the target convolutional neural network parameters through the PC software, and use python program to customize the arrangement of the parameters and image data to the memory parameter format under the target parallelism. Then, load the parameters and image data into the off-chip memory for the accelerator to call

(2) Under the control of the CPU, the image is scaled to adapt the size of the image to the input size of the convolutional neural network, and the calculation instructions of each convolutional layer are sent to the FPGA side through the AXI-lite bus. The sent instructions will be stored in the instruction buffer unit on the FPGA side. Finally, the DDR physical base address and start instruction are sent through the AXI-lite bus

(3) When the instruction decoding module sends the decoded calculation and memory access instructions of a certain convolutional layer to the global controller, the global controller will automatically call the computing kernel repeatedly and configure the instructions as control information required by the computing core, off-chip memory access, and on-chip cache

(4) RCFM (Read Configuration and Forwarding Module) will be the first to receive read commands from the global controller, and it will configure the configuration information required for the read operation according to these commands, including base address and transaction length. Then, this control information will be fine-grained and transmitted to RHPM, and then, RHPM will read off-chip parameters and data through the AXI bus

(5) After CFCC (computational flow and computation configuration controller) receives the reorganized data from the memory control system, it will first analyze the reorganized instructions and data. At

the same time, the corresponding channel information is generated, and the data are transmitted to the corresponding channel buffer through the AXI-STREAM bus

(6) The intermediate results of each output channel will be preprocessed and stored in the on-chip storage unit. When the output channel calculation under a certain degree of parallelism is completed, the single output channel will be divided into four channels for data-parallel processing, and finally, the intermediate processing results will be written back to the DDR

(7) The calculation of every layer will be completed through continuous iteration of (4-6). The final calculation result will be sent back to the CPU through the AXI-lite bus for processing

4.2. Instruction System. According to the characteristics of CNN calculation and parameter storage, this paper extracts two sets of instructions. The first set is a dynamic instruction set that can be dynamically configured after FPGA synthesis. It mainly deals with the adaptation of different convolutional neural network structures on the system architecture. The second set is the static instruction set used for the configuration before FPGA synthesis, which is used for the adaptation of the system architecture on the hardware platform with different resource constraints. The deep combination of the two can well solve the reconfigurable problem of the heterogeneous acceleration system of the convolutional neural network.

4.3. Coarse-Grained Dynamic Instruction. The coarse-grained dynamic instruction set is used to guide the calculation and memory access of a complete convolutional layer, and it is mainly oriented to the calculation and control of different convolutional neural network architectures under a fixed system architecture. The advantage of the coarse-grained dynamic instruction set is that the channel information is added to the instruction, and the calculation core can be repeatedly scheduled through the global controller to complete the convolutional layer calculation through one instruction configuration.

As shown in Figure 3, it is the coarse-grained dynamic instruction designed in this paper. [0-106] is used for memory access control, including AXI bus read and write base address and read and write length control. [89-106] is used for the automatic conversion of on-chip channels to achieve the purpose of controlling the base address of the memory access and the calculation mode. [107-117] is used for calculation control, used to control the convolution size, pooling step size, filling mode, and filling type.

4.4. Fine-Grained Dynamic Instruction. The fine-grained dynamic instruction set is used when the input feature map is too large, and a certain data block needs to be calculated and controlled finely. It is a supplement to the coarse-grained dynamic instruction set, which optimizes the adaptability of the same system architecture for different algorithm structures.

As shown in Figure 4, the fine-grained dynamic instruction set is mainly composed of four types of instructions: command type instructions, addressing, calculation, and control. Among them, there are three types of commands: parameter loading, data loading, and data writing back. [2:0] is responsible for the characterization. It mainly refines the process of automatic channel control of the coarse-grained dynamic instruction set into manual channel control. Addressing instructions are mainly composed of [76:3]. Calculation instructions are mainly composed of [85:77], including configuration instructions required for three calculations: convolution, pooling, and filling. Finally, there is a control instruction composed of [87:86], which mainly controls whether the output channel under the fixed output parallelism of a single data block has completed the calculation and whether all the data blocks constituting the input feature map under the fixed input parallelism have been input.

4.5. Static Instruction. The static instruction set design is mainly to solve the deployment problem of the system architecture on platforms with different resource constraints. The stage it guides is mainly the presynthesis stage, which is used for the scale expansion of the system architecture so that the system architecture can be adapted with hardware platforms with different resource constraints. A complete system architecture includes storage architecture and computing architecture. Therefore, the static instruction set designed in this paper also includes two parts of static instructions for calculation and storage, and these instructions exist in the parameter configuration file in the form of local parameters. The instruction bit width is 32 bits. Figure 5 shows the design of the static instruction set. Its core instructions are composed of six subinstructions. Among them, channel_in controls the hardware mapping dimension of the input channel in the system architecture, and channel_out controls the hardware extension of the output channel in the system architecture. On different hardware platforms, it is not only necessary to configure reasonable computing resources for a computing architecture but also to set up matching on-chip storage resources for it. Therefore, this paper configures four storage instructions for the allocation of on-chip storage resources. Among them, BRAM_size and BRAM_count are used for the size and quantity configuration of the intermediate result on-chip cache module, corresponding to the output parallelism. Line_size and line_count are used for the input feature map buffer configuration, corresponding to the input parallelism and the input feature map size.

4.6. Instruction Execution Module. After the dynamic instruction set is sent from the ARM to the FPGA through the AXI-GP port, it needs to be decoded and reorganized by a specific instruction receiving, decoding, and reassembly module on the FPGA. Figure 6 shows a block diagram of the instruction execution system, which is mainly composed of an instruction generation part and an instruction execution part. The entire instruction execution logic is as follows. First, INST_RECV communicates with CPU through AXI-

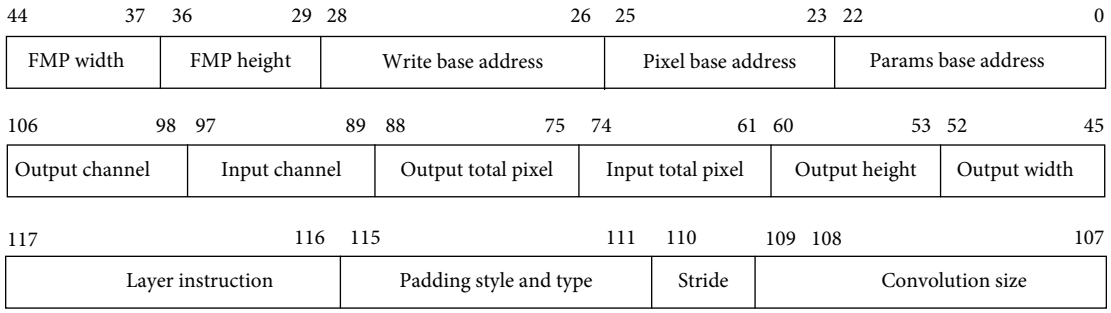

Figure 3: Coarse-grained dynamic instruction.

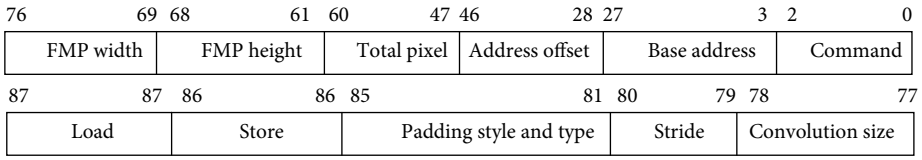

Figure 4: Fine-grained dynamic instruction.

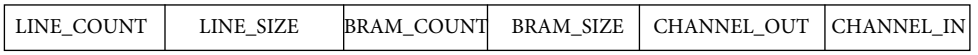

Figure 5: Static instruction.

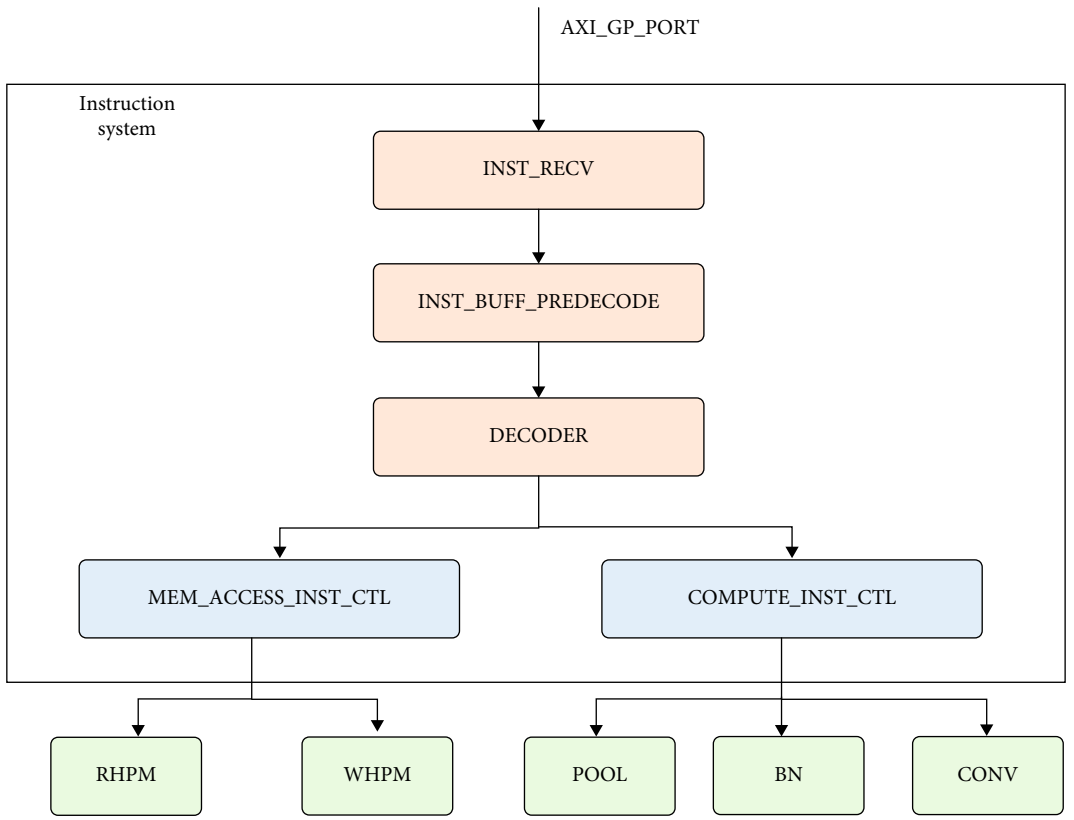

Figure 6: Instruction execution system.

lite and receives instruction data from the CPU. Then, through the command transmission bus, the command is buffered into the command buffer and predecoded. When the instruction buffer receives the instruction configuration of the last convolutional layer in the convolutional neural network, the module will start to send the predecoded instructions to the instruction decoder and decode them into commands that can be directly executed by each functional module. After decoding an instruction, it will send a feedback signal to the instruction buffer module and let the buffer send the next instruction that has been predecoded to the decoder. Then, the decoder sends memory access instructions such as addresses to the memory access instruction control unit and sends instructions related to the convolutional neural network calculation configuration to the calculation instruction control unit at the same time. Finally, the memory access instruction control unit and the calculation instruction control unit distribute different instructions to different calculation execution modules and memory access execution modules. The entire instruction parsing process is carried out in the form of a pipeline, and the next batch of instructions will be parsed within the time of this calculation, to achieve the purpose of time reuse.

4.7. Computing Architecture. To make full use of on-chip storage resources and computing resources and reduce the delay of computing critical paths as much as possible, it is necessary to optimize the design of the computing architecture. This paper proposes a fine-grained full-pipeline parallel computing architecture. Under this framework, this paper also designs a configurable convolution array for it to meet the requirements of computing performance and architectural expansion. In addition, the framework also provides ideas for parallel output data for postconvolution processing through the clever design of on-chip cache preprocessing, which improves the running speed of postconvolution processing operations such as pooling. The core of full-pipeline parallel computing is based on the full-pipeline computing architecture. By designing different scalable functional modules, the computational mapping of the convolutional neural network is completed. As shown in Figure 7, from the perspective of module design, the full-pipeline parallel computing core is mainly divided into six modules from left to right, which is responsible for data and parameter caching, convolution calculation, convolution intermediate result caching, batch normalization, activation, and pooling operations.

4.8. Convolution Array. The configurable convolution array is the core module in the full-pipeline parallel architecture, which is mainly used to undertake multidimensional convolution operations. The convolutional array designed in this paper makes full use of the structural characteristics of convolutional neural networks, multiplexing weight parameters, input feature maps, etc., using convolution kernel parallelism, output channel parallelism, and input channel parallelism to accelerate calculations, reducing the required bandwidth of computing system and improving the parallelism of computing. It has the characteristics of scalability, low critical path delay, and high throughput.

Figure 8 shows the overall data flow diagram of the configurable convolution calculation array, which is mainly composed of a configurable input line buffer module, a configurable weight buffer module, a configurable 2D-PE, and a configurable TPOCM. The line buffer module can complete data line buffering and data padding for input feature map of different sizes. Each data buffer can meet the buffering requirements of one input channel. As shown in Figure 8, under the premise of fixed hardware architecture, the pixel bus will be filled with pixel data required by all input channels, and at the same time, there will be several line buffer modules corresponding to the input channels mounted on the pixel bus. This paper also designs the weight buffer module corresponding to the pink part in Figure 8. This module is mainly composed of a weight distribution module and a 3D weight bus. Each weight distribution module corresponds to the weight distribution of an output channel. When the weight distribution module receives the channel selection signal, it will be enabled to receive the weight parameters that it deserves. At this point, all the parameters involved in the calculation are prepared, and the focus of the computing system operation is the convolution calculation array. The convolution calculation array is composed of a two-dimensional PE array composed of two-dimensional PE, which can complete all three-dimensional convolution operations. The design purpose is to calculate the multiple output channels of the convolutional neural network in parallel. Among them, the scale of the two-dimensional PE array can be flexibly expanded before synthesis through static instructions, and the design purpose of the two-dimensional PE is to independently complete the calculation of a single input channel of the convolutional neural network. At this point, the data preparation and calculation of the entire convolutional array can be carried out accurately. But due to the access bandwidth and memory access delay of the off-chip storage, the computing performance of the entire system will be severely hindered. This needs to consider the on-chip buffer of intermediate results in the output channel, so this paper designs TPOCM. This module is mainly responsible for providing temporary storage and accumulation of intermediate convolution results in the output channel. In addition, it also preprocesses the convolution results and provides parallel data streams for postconvolution processing.

To better explain the scheduling behavior of the convolutional array in the calculation of the convolutional layer, we assume that the hardware architecture configured by the static instruction is a convolution calculation array with 4 channels of input and 16 channels of output. Now we use the convolutional array under this parallelism to calculate the 16-channel input and 32-channel convolutional layer. The scheduling diagram of the convolutional array is shown in Figure 9. First, the input channel and output channel of the hardware architecture are smaller than the number of channels to be calculated in the convolutional layer, so multiple calls to the convolutional array are required to complete the calculation process of the entire convolutional layer. As shown in Figure 8, each blue part represents the system's scheduling of a convolutional array with 4 inputs and 16

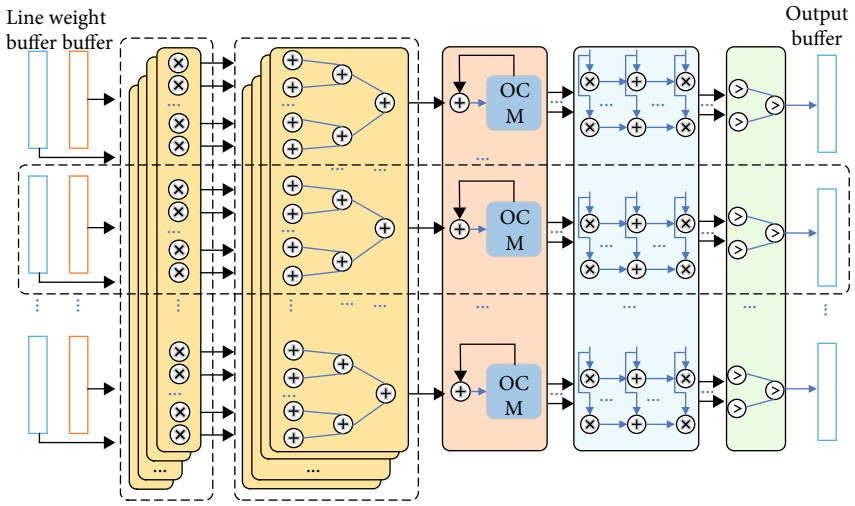

Figure 7: Multioutput channel computing core.

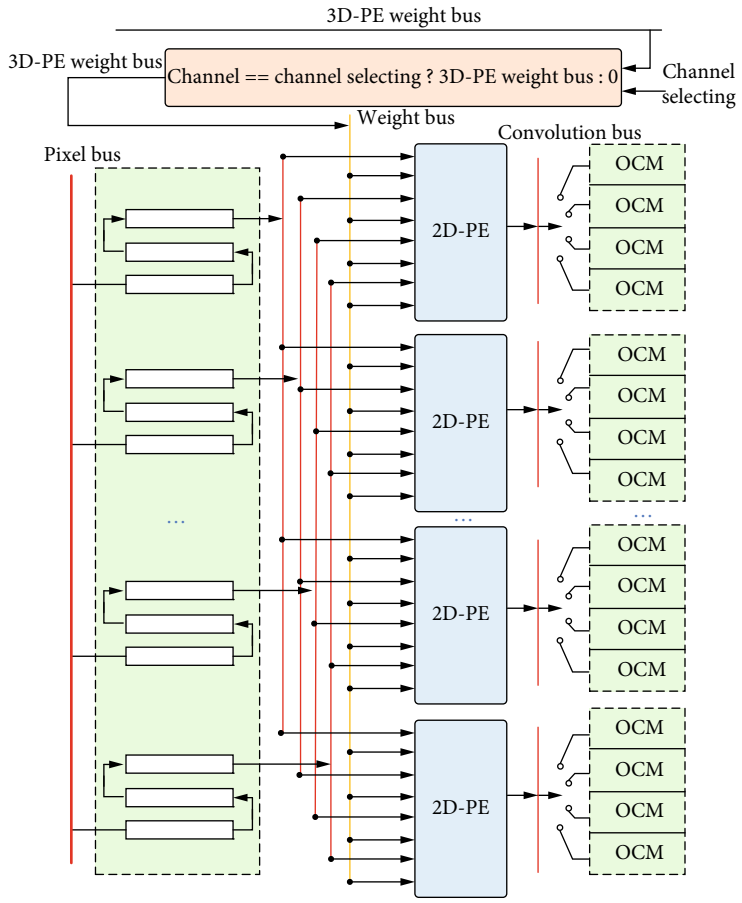

Figure 8: Schematic diagram of configurable convolution calculation array.

outputs. Each scheduling will first obtain the corresponding characteristic data and parameters from the off-chip DDR, then configure the calculation mode of the convolutional array, and finally send the parameters and data to the convolutional array in parallel for calculation. Since the input size of the convolutional layer is 16 inputs, the hardware architecture with 4 inputs must be repeatedly scheduled four times to complete the accumulation of 16 input channels, that is, the completion of 16 outputs at one time. Also, because the output size of the convolutional layer is 32 channels, and the output size of the convolutional array is 16, it is necessary to call the convolutional array with 4 inputs and 16 outputs 8 times to complete the calculation requirements of the convolutional layer. The 4 independent blue schedules

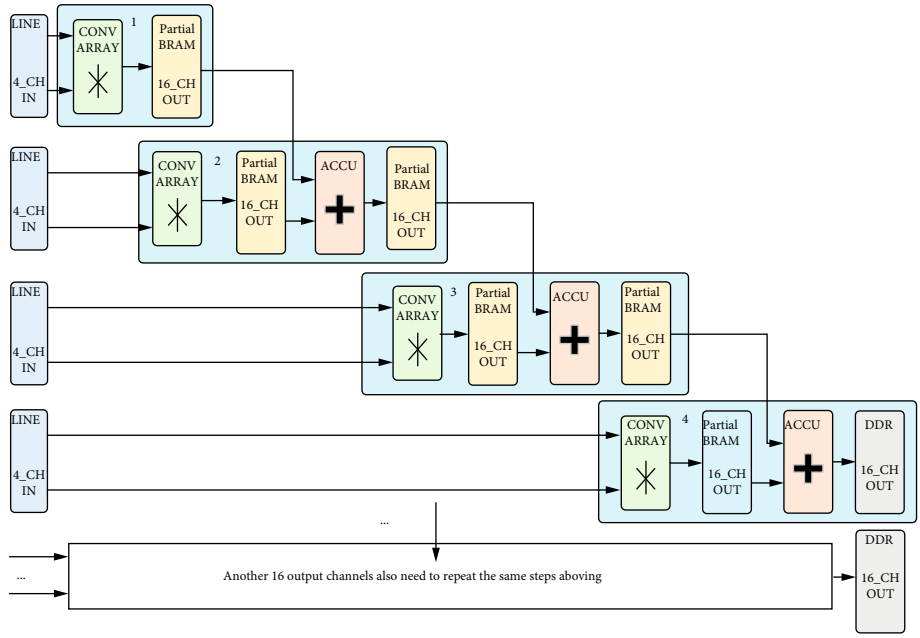

Figure 9: Scheduling graph of convolutional array.

shown in the figure form independent 16-input-16-output channel calculations, and the abovementioned 16-input-32-output convolutional layer calculation can be completed by repeating the scheduling process. At this point, the convolutional layer calculation can be completed by scheduling the convolutional array. If you configure different convolutional layers through dynamic instructions as shown in Figure 3, then the fixed convolutional array structure can completely handle different structures of CNN. The following is an introduction to the ingenious design ideas of the cache and calculation module from a more detailed perspective.

4.9. PE. The configurable PE is the most basic arithmetic unit that composes the convolution array. After optimizing design, the PE unit designed in this paper adopts a configurable structure of parallel multiplication and pipeline addition, which can independently complete the convolution operation with convolution kernel size of 3 or 1.

Each PE contains nine multipliers and eight adders, and its pipeline has a passing time of five clock cycles, after which each clock cycle can produce a convolution result. Compared with the similar streaming PE structure, it has stronger versatility and higher utilization of resources and can better achieve different computing goals by reusing existing resources, which is reflected in the resource-constrained platform more obvious. As shown in Figure 10, the figure shows a data flow diagram of a convolutional neural network with a configured convolution size of $3 * 3$. The calculation logic shown in the figure is as follows: Firstly, the master control unit will receive an instruction from the CCFM to configure the size of the convolution calculation which will be completed by the PE. Secondly, the weight buffer in the figure will obtain nine weight parameters belonging to the specific input channel from the weight bus, and the pixel window bus provides pixel data for nine parallel DSP multipliers.

4.10. 2D-PE. 2D-PE array is an important component of the convolutional array. This module can independently complete the calculation of an output channel, which means if time efficiency is not considered, it can undertake all the convolution calculations of the convolutional neural network.

Figure 11 shows the data flow diagram of a configurable two-dimensional PE array with an input parallelism of 4. From a structural point of view, it is still mainly composed of the input buffer, output intermediate result buffer, and calculation module. For the 2D-PE array, the main focus is on the structure of the calculation part in the box shown in Figure 11. The number of PEs in the figure is 4, which means the module has four-input parallelism and can complete the partial sum of one output channel in one round of calculation. Assuming that the input channel of a certain convolutional layer is N, then the 2D-PE array needs to be repeatedly called for N/4 rounds to complete the calculation of all the sums of one output channel of the convolutional layer. The input parallelism of the hardware architecture in this design can be configured based on on-chip resources through static instructions. Therefore, the number of PEs in a single 2D-PE array is configurable, and the corresponding addition operations are also scalable. As for the 2D-PE weight bus, the weight bus is filled with weight parameter groups corresponding to the number of PEs. They are arranged on the bus in the order from low to high with data bit width as the unit so that each PE unit can obtain weight parameters from the bus in parallel, which meets the requirements of the PE internal calculation logic. So to meet the requirements of such parameter acquisition, this paper designs a parameter acquisition system as shown in Figure 12. The design thinking is briefly introduced below.

This paper accesses the DDR of CPU through the AXI bus. Due to the limitation of the number of on-chip high-

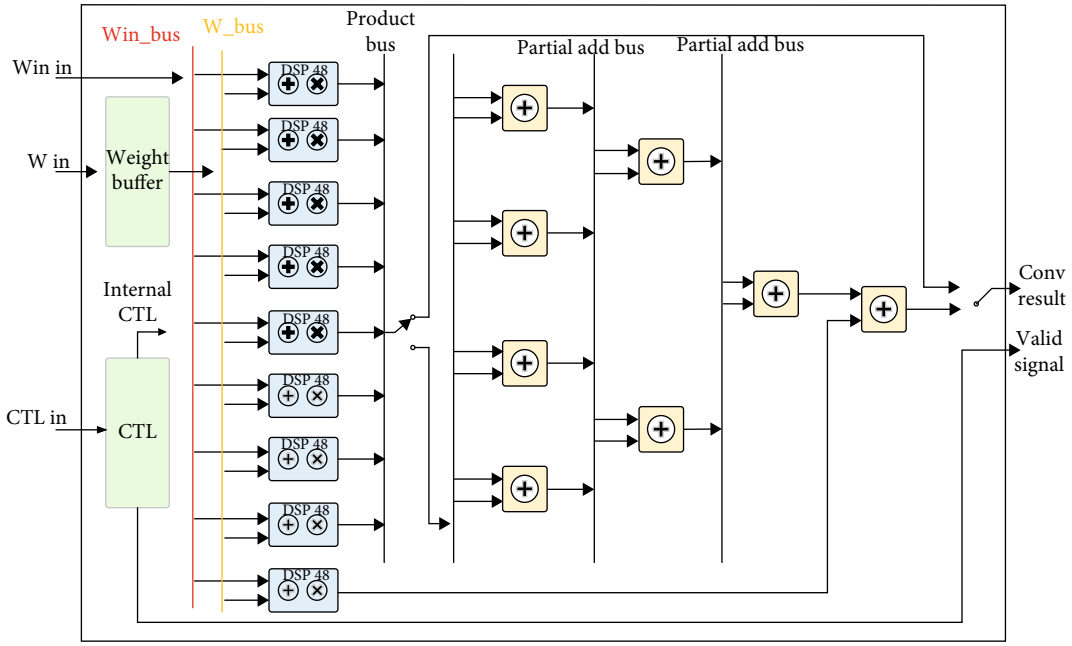

Figure 10: Internal structure of configurable PE.

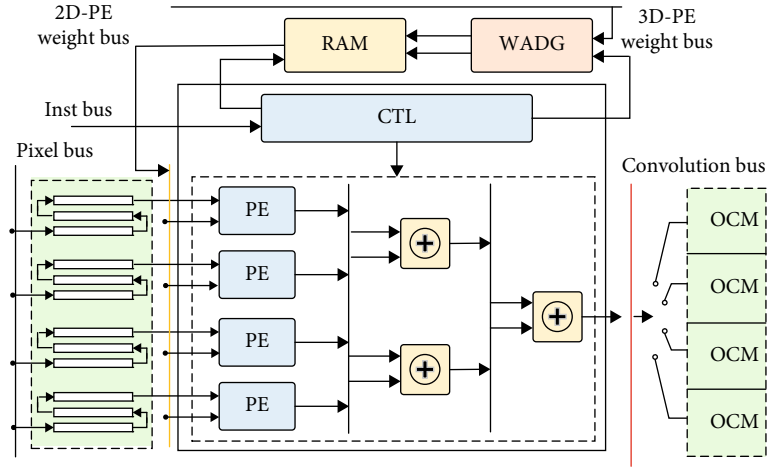

Figure 11: Schematic diagram of 2D-PE array.

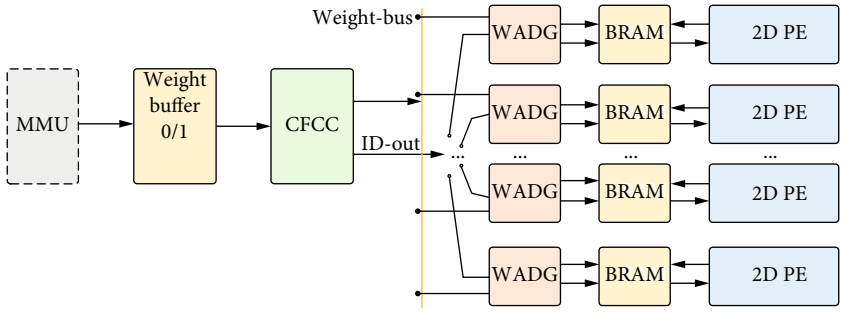

Figure 12: Design diagram of weight distribution module.

speed access interfaces and bandwidth required to access the DDR, interface resources need to be multiplexed. However, the weight parameters of the convolutional neural network need to be sent to the weight buffers corresponding to different input and output channels. Therefore, this paper designs a special parameter distribution mechanism that can accurately send the parameter data from the MMU to the corresponding weight buffer. Figure 12 shows the parameter distribution system designed in this paper. The MMU will store the data obtained from the memory in the weight buffer in the form of a ping-pong operation, and then, the CFCC module will read the parameters in the weight buffer and generate the ID of the output channel corresponding to the parameter. Furthermore, the system will strobe the output channel corresponding to the ID and store weights in the on-chip BRAM corresponding to each input channel through the weight address generating unit.

4.11. Postconvolution Processing. The postconvolution processing module is mainly responsible for processing the convolution results of part of the output channels of a certain convolution layer completed by the convolution array. The overall schematic diagram is shown in Figure 13. The postconvolution processing module is mainly composed of TPOCM (partial sum and preprocessing on-chip memory module), AP (activation and pooling module), and postprocessing module.

4.12. TPOCM. TPOCM is mainly responsible for two parts of work. The first part is the temporary storage of the intermediate results in the output channel of the convolutional array and the accumulation for the output channel. The convolution results are preprocessing during the temporary storage to facilitate subsequent reading in parallel. The second part of the work is mainly to read the on-chip output feature map of the completed operation according to the instruction configuration requirements and provide a high-speed parallel data stream for activation and pooling operations.

As shown in Figure 14, the modules responsible for these two parts are mainly divided into two parts: the control module and the on-chip storage block. First, focusing on the control module, from the perspective of composition, the control module is mainly composed of CORDG (image coordinate generation module), TSADG (data on-chip temporary storage address generation), SAMPLE (instruction sampling module), PPADG (preprocessing address generation module), and timing alignment and accumulation modules. The workflow is as follows: First, the sample module will sample the instructions distributed from the CFCC, mainly to obtain the feature map size, and distribute the instruction information to the CORDG module and the PPADG module. The CORDG module is mainly responsible for the calculation of pixel coordinates in the feature map and transmits the calculated pixel coordinates to the TSADG module. The TSADG module divides the feature map into blocks with four areas as a basic unit and generates the same addresses according to the odd and even rows (columns) of the four-neighborhood pixels after the block. At this time, the global controller will determine whether the calculation

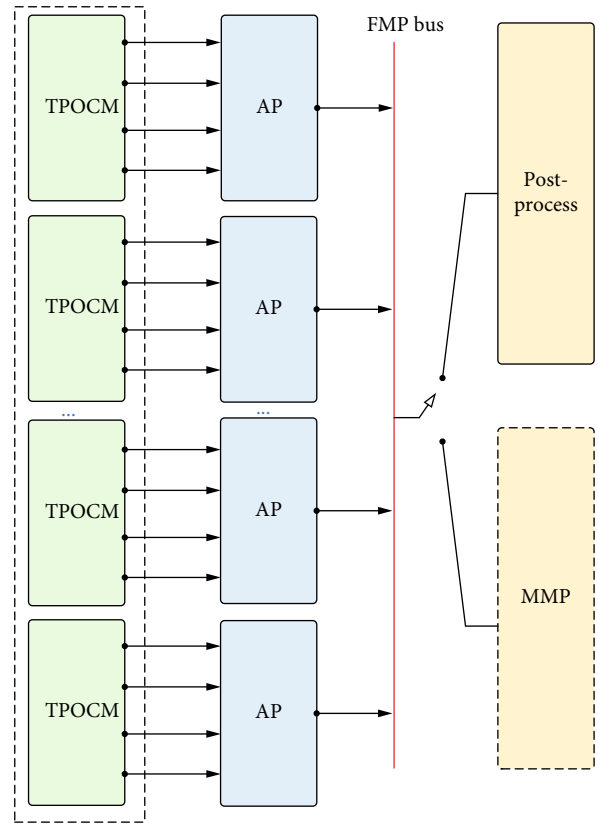

Figure 13: Schematic diagram of convolution postprocessing module.

of the output channel of the current convolutional layer is completed. If it is completed, the pixels with the same parity row and column in each of the four areas are written into the corresponding on-chip BRAM according to the address and then handed over to the enable PPADG for readout operation. The pixels in the four areas of the corresponding BRAM are output to the AP module in parallel for subsequent processing. If not completed, the result is first output to the accumulator, and the intermediate result of the corresponding output channel is read from the BRAM and transmitted to the other input port of the accumulator to complete the update of the intermediate result. In summary, the most critical idea of the entire control module and cache structure is to preprocess the feature map data of the four fields to achieve the purpose of accelerating the processing parallelism of the AP module.

4.13. AP. AP is the main calculation module for postconvolution processing. It is mainly responsible for the activation function and pooling operation. Its basic structure is shown in Figure 15, using a full pipeline structure. The inside is mainly composed of the control unit CTL, the activation calculation part, and the configurable pooling calculation part; the outside is mainly composed of the TPOCM module.

As shown in Figure 16, the workflow is as follows: First, the internal control module receives the control instructions and parameters from the CCFC and generates

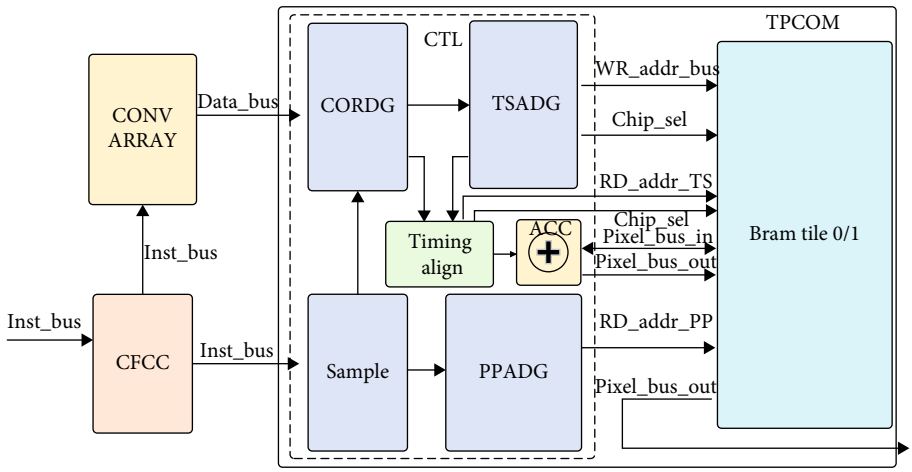

Figure 14: Schematic diagram of TPOCM preprocessing module.

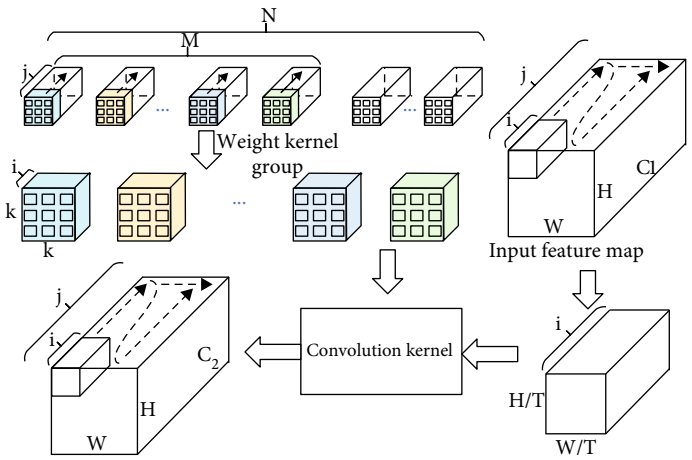

Figure 15: Division diagram of parameters and weights.

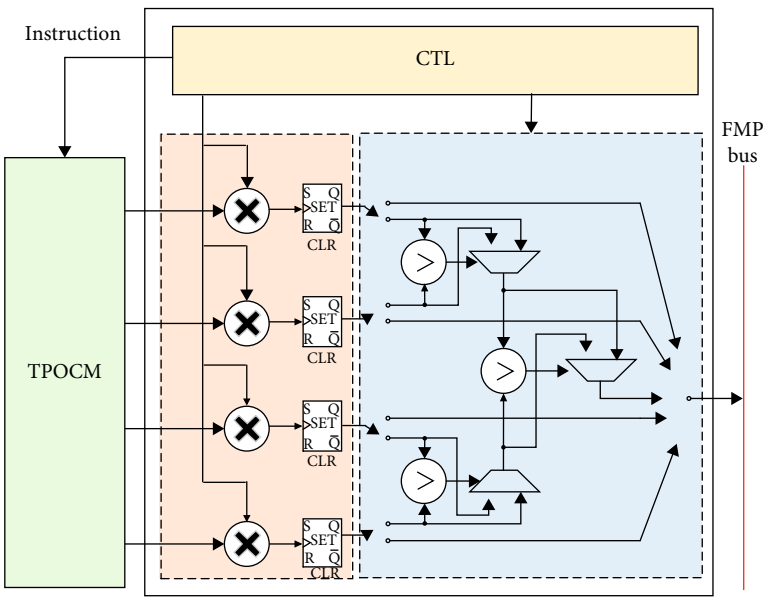

Figure 16: AP module diagram.

the corresponding pooling stride size control instructions to control the data address generation and internal pooling after the internal convolution of the TPOCM module. Then, the system will change the stride length control and at the same time put the received parameters into the corresponding multiplier input. The TPOCM module will pass the output data corresponding to the stride length to AP in parallel according to the address, block selection, and chip selection signals. After the AP undergoes activation calculation, the corresponding result will be bridged or sent to the comparator according to the stride size control signal for pooling of different stride lengths. Finally, AP finishes the postprocessing of convolution.

4.14. Memory Management System. The memory management system provides data and parameters for the entire heterogeneous acceleration system. The memory management system is a collection of a series of operations and methods related to the arrangement of data and parameters, address division, memory division, arrangement, and address mapping access. So the design of the entire system includes front-end planning and design and back-end execution unit design. Specifically, first of all, the memory management system designed in this paper has a methodology for dividing parameters such as data and weights in memory. According to this methodology, this paper designs a set of python codes to rearrange the data and parameters into a form suitable for the hardware architecture. Secondly, based on the memory mapping theory, this paper designs a complete set of memory management execution modules for FPGA. Compared with the traditional DMA IP, this module has a greater degree of freedom and is compatible with different versions of the AXI bus. These two parts are introduced in detail below.

4.15. Memory Division and Address Design. The addressing method of heterogeneous systems is unified addressing, that is, all IO registers and main memory units are unified into the memory system. This paper uses this technology to divide the memory space of the entire heterogeneous system into the structure shown in Table 1 which is mainly divided into six areas, namely, the parameter storage area, the image storage area 1/2, the convolutional layer buffer area 1/2, and IP register area. Among them, the parameter storage area is mainly used to store weights, batch normalization parameters, etc. The image storage area is mainly used to store image data from the camera or PC, and ping-pong operations are running at the same time; the convolutional layer buffer is mainly stored outputs results of the middle layer of the convolution accelerator from the FPGA. The IP register area mainly corresponds to the internal configuration registers and information feedback registers mapped by the AXI-GP interface of the convolution accelerator. In addition, this paper also designs the memory address corresponding to the memory space of the heterogeneous accelerator according to the divided memory system.

4.16. Data and Parameter Division. How to propose a set of workflows for the division of data and parameters based on the resource changes of different platforms is very critical for the entire system. Starting from the bandwidth limitations of heterogeneous platforms, the following discusses the establishment of the overall workflow.

Table 1: Memory space division and address range.

Storage area	Start address	End address	Storage space
Parameter storage	0x1000000	0x20400000	500 MB
Image storage 1	0x30000000	0x30FFFFFF	16 MB
Image storage 2	0x31000000	0x31FFFFFF	16 MB
Convolution layer cache 1	0x32000000	0x32FFFFFF	16 MB
Convolution layer cache 2	0x33000000	0x33FFFFFF	16 MB
IP register	0x40000000	0x7FFFFFFF	1024 MB

This paper takes XC7Z100 as an example and uses static instructions to configure the hardware architecture of 4IN-32OUT. The architecture under this parallelism is the result of fully considering the DSP resources and on-chip storage resources under a specific heterogeneous platform. This heterogeneous platform is equipped with 4 DDR3 memory cores. The data bit width of a single memory core is 16 bits, two of which are connected to the PS side, and the other two are connected to the PL. The maximum data frequency of the memory is 1066 MHz, so the maximum data bandwidth can be $1066 * 32 * 106$ bit/s = 4.164 GB/s. Correspondingly, the highest working frequency of the AXI bus of this heterogeneous platform is 250 MHz, so for the four-way AXI-HP interface, the overall access bandwidth can reach $4 * 250 * 64$ bit/s = 7.8125 GB/s. Therefore, it can be determined that the access bandwidth of the AXI bus can completely cover the physical access bandwidth of DDR. Returning to the heterogeneous acceleration architecture, the memory access structure originally designed in this paper contains three AXI-HP interfaces, which are responsible for the data path, parameter path, and write-back path. The bit width is 64 bits and works at 210 MHz. When the three channels work together, the total access bandwidth is $3 * 210 * 64$ bit/s = 4.921875 GB/s at this time. This access bandwidth has exceeded the maximum access bandwidth of PS-DDR, so this solution is not advisable. Through the research and improvement of the scheme, we find that the access mechanism of the parameter path is prefetching, that is, the next access of the parameter path overlaps with the current data path access. The period of the data path is much longer than that of the parameter path, so the access frequency of the parameter path can be reduced to 100 MHz. The overall access bandwidth obtained at this time is reduced to $2 * 210 * 64$ bit/s + $100 * 64$ bit/s = 4.0625 GB/s. Now, the overall access bandwidth is less than the upper limit of the PS-DDR access bandwidth. If the subsequent ARM's demand for PS-DDR bandwidth increases, the AXI-HP interface bandwidth may be compressed. Since the data path provides real-time computing data streams, its computing bandwidth must be consistent with the overall system bandwidth, so at

this time, the standard can only be achieved by reducing the bandwidth of the write-back path. However, this processing may cause the next batch of calculation results to overwrite the previous batch of calculation results. If this happens, we can also use ping-pong operations to improve this situation.

Under the fixed 4IN-32OUT hardware architecture, the limitation of access bandwidth was discussed, and the overall access bandwidth was finally limited to 4.0625 GB/s. In fact, under this hypothetical hardware architecture, the form of parameter access is also fixed. Taking the weight parameter as an example, in the calculation process of the convolutional layer, $4 * 32 * 9 = 1152$ weight parameters are required each time, so the space of the on-chip ping-pong buffer of the weight parameters is predetermined. If the parameter quantization bit width is 16 bits, the storage size occupied is 18 Kb. Specifically, as shown in Figure 15, what is described is the data division method of a certain convolutional layer. There are a total of N convolution kernel groups shown in the figure. Each convolution kernel group corresponds to the calculation parameters required by an output channel. Each convolution kernel group is composed of j convolution kernels, which also means j input channel corresponds to it. When the input channel of the computing architecture is i and the output channel is M, all weight parameters required for each round of calculation are shown in the colored squares shown in Figure 15. These parameter groups will be obtained by the acceleration peripheral on the FPGA through the AXI-HP interface and sent to the designated computing core register. When this round of calculation is completed, the global controller will change the interactive address to obtain the operation parameters of the next i input channel in the M convolution kernel groups, until the system finish traversal in units of i for all the convolution kernel groups of the M channels. Then, through address conversion, the traversal target is converted to the next set of M output channels, until the traversal of N output channels finishes in units of M. At this point, the acquisition of all the parameters required for the calculation of the convolutional layer has also been completed. Then, according to the above acquisition rhythm, in the specific off-chip memory, the weight parameters need to be arranged according to the above rules. Therefore, this paper designs a special python program to solve the arrangement of parameters under different input and output channels.

The above is mainly the analysis of parameter layout and parameter acquisition. For the data path, the demand for the on-chip cache is mainly concentrated on the input channel data buffer and the output channel convolution calculation intermediate result buffer under a certain hardware architecture. For example, the 64-bit AXI-HP interface aforementioned works at 200 MHz. If the input parallelism is increased, it means that this bit width cannot meet the parallel input of characteristic data at 200 MHz, and the bandwidth requirements will be greatly increased. Therefore, the method selected in this paper is to give priority to the increase of output parallelism. Limited by the size of the input feature map, especially the large-size feature maps of the first few layers, we cannot cache the complete calculation results on the chip. Assuming that our input feature map is $416 * 416$ and the quantization bit width is 16 bits, then the storage of a single output channel requires 2704 Kb. Then, if the 32 output channels mentioned above are implemented, a total of 86528 Kb is required, which is much larger than the 27180 Kb on-chip storage resources of the XC7Z100 heterogeneous platform. Therefore, when facing large-size input feature maps, the feature maps cannot be input into the entire system as a whole. As shown in Figure 17, we split the input feature map with T as the block constant and split it into input data blocks with H/T as height, W/T as width, and i as the input depth, which we call the 2D block division of input data. Similarly, the whole process of traversing the entire input 3D feature map is shown in the input feature map in Figure 15, first traversing the depth direction, then traversing the horizontal direction, and finally traversing the vertical direction.

4.17. MMU. The previous section mainly discussed the division of the memory system and the design of data arrangement. For the entire heterogeneous acceleration system, these designs also require a specific execution system. Therefore, according to the protocol requirements of the AXI bus and the characteristics of the memory system of the heterogeneous acceleration system, this paper designs a memory mapping unit as shown in Figure 17, which is mainly responsible for the mergence of system memory access and on-chip communication data. According to Figure 17, the entire memory mapping unit is mainly composed of a configuration and data forwarding unit (CFU) and a high-performance memory access unit (HPM). The CFU is mainly responsible for the coarse-grained configuration of reading and writing tasks, fine-grained memory access information reorganization, and on-chip communication mergence and forwarding. HPM is the main access terminal for accelerator access to DDR, which is mainly responsible for the configuration of fine-grained burst transmission, data receiving, and sending buffer. As shown in Figure 17, buffers for different purposes are equipped with a set of memory mapping units for active access to DDR and internal command configuration, as long as the combined access bandwidth of these buffers is not greater than the highest transmission bandwidth of DDR.

As shown in Figure 18, we take the read memory mapping unit as an example. The workflow is as follows: First, GBCTL (global controller) will fetch the memory access addresses and related instructions designed according to the memory division from the instruction buffer and submit them to the RCFU. RCFU adjusts the base address and burst information according to the task instruction information and, in the meanwhile, reorganizes them and generates address information used by RHPM. RHPM will receive the base address and burst information under the transmission task and automatically adjust the burst information to change the access address to complete a burst of the transmission task. At the same time, RHPM will forward the received data to RCFU, and the handshake signal will enable RCFU to readjust the task information for the next burst information configuration. In addition, the RCFU will

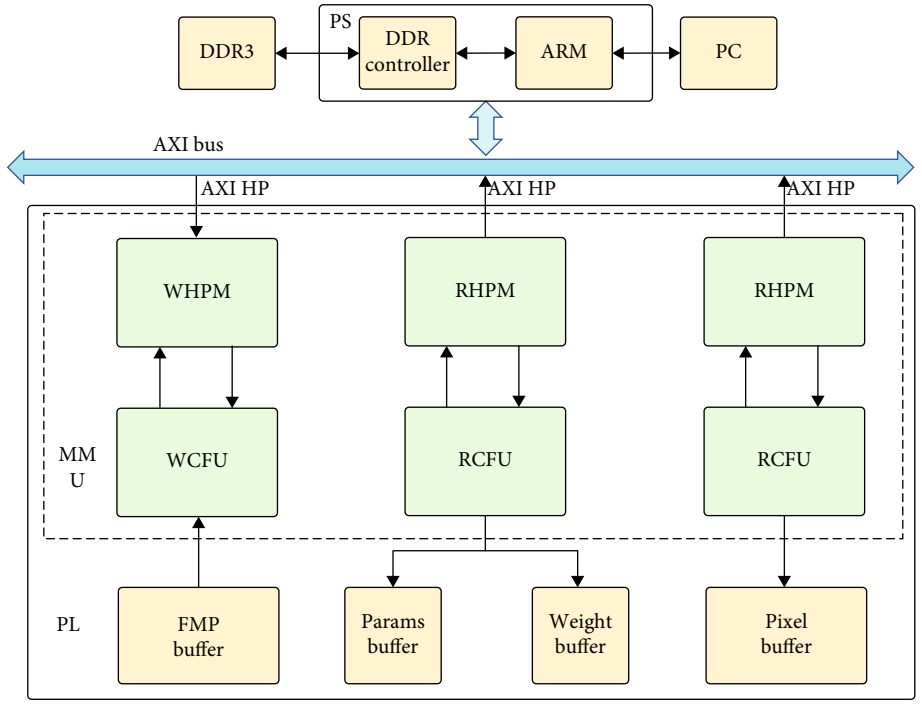

Figure 17: Schematic diagram of memory mapping unit.

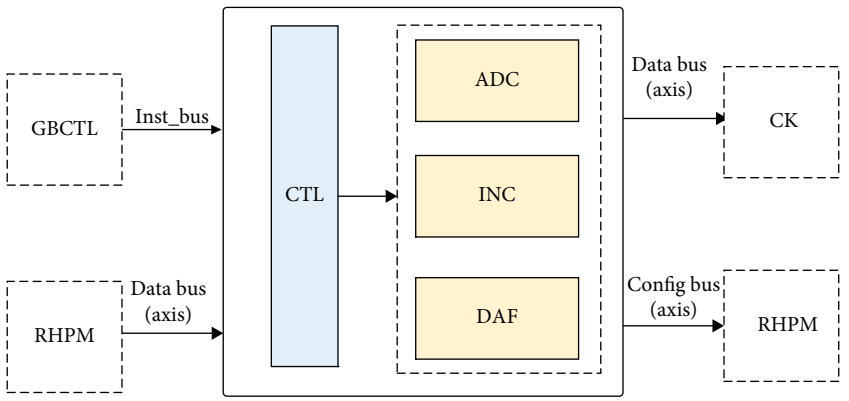

Figure 18: Internal schematic diagram of RCFU.

reorganize and merge the received data with the calculation instruction information and finally forward it to the computing core.

5. Results and Discussion

The convolutional neural network workflow proposed in this paper is oriented to the classical convolutional neural network at this stage. Each component subsystem has the characteristics of configurable instructions and reconfigurable while system maintaining high energy efficiency and computational density. In addition, for relatively niche networks, the accelerator can quickly meet network computing requirements by adjusting the corresponding subsystem architecture parameters or cascading new computing modules, which also reflects the advantages of the full pipeline architecture.

We choose the classic YOLO series of algorithms to verify the core of the universal convolutional neural network acceleration. This is related to the following research in this paper. This research is dedicated to applying this general convolutional neural network acceleration core to specific real-time engineering processing, so the YOLO series of algorithms with high real-time performance are selected for the verification of the acceleration core. After a joint evaluation of the amount of calculation and existing heterogeneous platforms, this paper finally chose YOLOv2-tiny and YOLOv3-tiny as the verification algorithms of the general convolutional neural network acceleration core and finally adapted them on the XC7Z100. Another reason for choosing the YOLO algorithm is that its backbone network

is a very classic convolutional neural network at this stage, and its convolution size, pooling stride size, and other characteristic parameters are typical representatives of the current convolutional neural network, which can be very perfect algorithm to verify the feasibility of the universal acceleration core in this paper.

5.1. Scalability and Reconfiguration. First, let us analyze the scalability of the convolutional neural network acceleration computing core. As shown in Table 2, we select YOLOv2-tiny to perform reconstruction experiments under three different acceleration architectures. When the algorithm does not require much computing performance, we can select a reconfigurable platform with few resources and low cost; when the on-chip storage and computing resources of the reconfigurable platform are abundant, we can add PE and 2D-PE to achieve higher accelerator computing power. Making full use of the scalable model of this static architecture also allows us to flexibly use hardware resources to replace accelerator performance. Specifically, as shown in Table 2, driven by different architecture parallelism, it shows the consumption of on-chip storage and computing resources under different PE-2DPE configurations, as well as the time-consuming and performance of the accelerator during the inference. These results are obtained after we deploy the accelerator on the XC7Z100, which has 277,000 LUTs, 755 BRAMs, and 2020 DSPs. It can be observed from this table that DSP resource consumption increases fastest with the expansion of PE-2DPE and is also the closest to the upper limit of the device. This is because DSP resources directly affect computing performance, and high computing performance must have high DSP consumption. On-chip BRAM mainly limits the complexity of instruction configuration. Therefore, in the case of low BRAM consumption, it means that fine-grained dynamic instructions will be configured complexly, and the scheduling of the memory management system will be relatively frequent. When the PE-2DPE architecture configuration reaches 4IN-32OUT, the PL inference speed of the entire accelerator can reach 31.27 FPS. In the case of adding part of the postprocessing on the PS side, this maximum throughput will be reduced to a certain extent, which is caused by the data interaction between the PS and the PL and the computing architecture of the PS.

From the perspective of dynamic reconfiguration, the accelerator designed in this paper can adapt to different convolutional neural network algorithm structures through the configuration of dynamic instructions. This paper selects YOLOv3-tiny for comparison experiments. After adding the upsampling module, the reconfigurable heterogeneous accelerator is fully capable of YOLOv3-tiny's computing requirements. In addition, the accelerator can also be adapted to the calculation of some convolutional neural networks such as VGG through fine-tuning, which greatly decreases the period for heterogeneous accelerators to adapt to different algorithm structures. As shown in Table 3, when the architecture is fixed at 4IN-16OUT, the frame rate of YOLOv2-tiny can reach 20.44 FPS, while the frame rate of YOLOv3-tiny is 21.29 FPS. This result is still the result of serializing YOLOv3-tiny's prediction branch. Later, some on-chip resources will be used to implement the calculation of YOLOv3-tiny's prediction branch for parallel computing to further optimize the throughput of heterogeneous accelerators.

5.2. Result. This paper uses Verilog to model and implement the designed accelerator architecture and uses Vivado 2017.4 for comprehensive layout and wiring. This heterogeneous device also contains a dual-core ARM core. The finally demonstrated architecture is a 4-16 parallel architecture, which runs on the PL side.

Table 4 shows the resource consumption table of the accelerator's computing core on the PL. Table 4 divides the computing core into a computing part, a cache part, a control part, and a memory management part. As shown in Table 4, the convolutional array in the calculation part consumes about 80% of the accelerator DSP resources. This is because the main calculations in the convolutional neural network are concentrated in the convolutional layer, which leads to the high computing density in the computing core. In addition, the TPOCM module in the on-chip cache part consumes 128 BRAM resources. This is because a TPOCM module needs to store the intermediate calculation results of the entire feature map to achieve the goal of increasing the calculation density. In addition, various subsystem controllers mainly consume LUT resources, which is the result of a large number of state machines for scheduling. The global controller also consumes some BRAM resources for buffering calculation parameters. The memory management system mainly consumes LUT and BRAM resources, because the memory management system involves protocol conversion and data access buffering.

In addition, the heterogeneous accelerator under the 4-16 architecture will eventually run on the XC7Z100 at a frequency of 210 MHz. At this frequency, Vivado estimates the power consumption of the accelerator computing core to be 4.515 W, of which static power consumption accounts for 0.292 W, and dynamic power consumption accounts for 4.222 W. Under the operating frequency of 210 MHz, the whole computing performance of the heterogeneous accelerator under the 4-16 architecture implemented in this paper under ideal conditions is 98.01 GOP/s. Table 5 shows the various parameters and inference performance of each convolutional layer of the YOLOv3-tiny network. Through observation, it can be found that a large increase in convolution depth will cause a large increase in inference time. This is because a substantial increase in the convolution depth will not only increase the amount of calculation but will also intensify the repeated scheduling of the accelerator's computing core. The accelerator implemented in this paper is based on the standard AXI protocol for communication to take universality and rapid scalability into account. Therefore, a part of the time will be spent on data handshaking and transmission. For the initial large-scale feature map input, there is no significant inference delay for the accelerator. This is because the on-chip BRAM resources of XC7Z100 are relatively abundant. Therefore, this paper uses the static reconstruction function of the heterogeneous accelerator to increase the resources of the TPCOM module,

Table 2: Analysis of system resources and performance under different architecture reconfiguration.

Architecture	Slice LUT	Slice	LUT logic	DRAM	BRAM	DSPs	Processing time (ms)	FPS
4-4	21757	8632	20285	1472	59	181	149.94	5.67
4-16	64391	25105	59034	5357	155	673	48.92	20.03
4-32	120593	46332	110119	10474	283	1329	31.98	31.27

Table 3: System performance analysis under different convolutional neural network algorithm structures.

CNN	Architecture	Input size	Processing time (ms)	FPS
YOLOv2-tiny	4-16	416 ∗ 416	48.92	20.44
YOLOv3-tiny	4-16	416 ∗ 416	46.95	21.29

thereby greatly reducing the data transmission time consumption for the block calculation of large feature maps. At the same time, since the number of blocks is reduced, fine-grained dynamic instructions do not require a large amount of configuration, which also simplifies the complexity of instruction configuration.

5.3. Accelerator Optimization Strategy and Gain Analysis. In the research of digital high-speed circuits, performance, power consumption, area, and generality are the four key indicators. The relationship among the four is mutually restricted. People often hope to find a balance among the four in the corresponding application scenarios. Therefore, the design and optimization of hardware accelerators are comprehensive, including critical path optimization of digital circuit design, optimization of computing resource utilization, optimization of memory access efficiency of memory resources, optimization of circuit generality, and instruction execution efficiency. The accelerators shown in this paper have been optimized in various degrees in the above aspects, and what they ultimately reflect is the increase in computing density, energy efficiency ratio, and generality.

Specifically, first, from the perspective of critical path optimization, the shortening of the critical path is a key condition for the increase of the main frequency of the digital high-speed circuit, and the increase of the main frequency means the increase of the calculation bandwidth of the system. This paper mainly optimizes the main frequency of digital design from two aspects. First of all, based on the timing report, this paper constantly adjusts the length of the critical path and finally limits the critical path to the length of a single DSP hard core. Secondly, in further timing analysis, we found that the delay of the net was too large. After research, this was caused by the excessive fan-out of the data bus. Therefore, we used the method of logic replication to shorten the net delay and further shorten the critical path length. In the end, the main frequency is maintained at a higher level. Second, from the perspective of computing resource utilization and computing efficiency, this paper fully incorporates the classic ideas of digital hardware architecture design such as parallel, pipelining, time-sharing multiplexing, and folding in the design, which improves the utilization of computing resources and computing efficiency. Specifically, as shown in Figure 7, we have added a parallel multiplication and pipeline addition calculation architecture to the convolution calculation, which improves the passing time of the calculation path and reduces the critical path time. In the subsequent postconvolution processing, the circuit is universally controlled, so that different functions of the same circuit are scheduled at different times, which improves the multiplexing of hardware resources. Third, in terms of memory access efficiency, this paper makes full use of the low latency and low power consumption characteristics of on-chip storage resources and designs the corresponding on-chip storage module TPOCM. On the one hand, the on-chip cache module reduces the access requirements of the entire accelerator for off-chip storage, which improves the memory access efficiency of intermediate results and reduces power consumption. On the other hand, through ping-pong operation, the computing bandwidth of the overall computing core is increased. And at the same time, a high-speed parallel data stream is provided for postconvolution processing. Fourth, this article reduces the redundancy of the original DMA scheme by customizing the memory management unit for the accelerator, which reduces the on-chip area and improves the overall off-chip memory access efficiency. The direct effect of these four-point optimizations is the comprehensive improvement of calculation density and energy efficiency ratio.

Because the RTL-level modeling of the DCNN hardware acceleration system is very complicated, once the system is determined, the cost of system RTL-level changes is very large. In addition, the influence of design and optimization on the acceleration result is comprehensive, so the final acceleration system performance and other evaluation index improvement factors are relatively complicated. Therefore, for hardware acceleration systems, the differences between different acceleration systems and the corresponding benefits are difficult to accurately locate and limit, and it is more to compare the overall performance indicators of different acceleration systems. That is, this paper is difficult to directly give precise ablation experiments like the research in the field of deep learning. However, it is critical to show the performance benefits of modules that are different from existing work. Therefore, we carried out the following two tasks. From the perspectives of computing resources, main frequency, on-chip storage resources, and computing bit width, we selected work very close to this article (if the optimized main frequency has a certain multiple difference, we

Table 4: Accelerator resource consumption statistics under 4-16 architecture.

Name	Slice LUTs	Slice registers	Slice	LUT logic	DRAM	BRAM	DSPs
CONV array	24391	43515	15032	20446	3945	0	540
AP array	27065	23280	9106	26495	570	0	80
PE	210	530	218	144	66	0	9
2D-PE	1693	3090	979	1415	278	0	36
TPOCM	1584	2208	1193	1584	0	128	0
Line	555	1573	467	555	0	4	0
TPOCM CTL	2262	2312	1332	1947	315	0	15
AP CTL	1993	242	909	1993	0	0	0
CONV array CTL	304	381	218	304	0	0	0
Global CTL	893	1060	376	893	0	8	0
MEM management	1221	1377	587	1214	7	8	0
Total	64391	81460	25105	59043	5357	155	673

Table 5: Performance statistics of YOLOv3-tiny each layer under 4-16 architecture.

Input FMP	Output FMP	CONV size	Ops	Processing time	Performance (GOP/s)
416 ∗ 416 ∗ 3	208 ∗ 208 ∗ 16	3	74760192	1.6 ms	93.45
208 ∗ 208 ∗ 16	104 ∗ 104 ∗ 32	3	199360512	2.29 ms	174.11
104 ∗ 104 ∗ 32	52 ∗ 52 ∗ 64	3	199360512	2.36 ms	168.95
52 ∗ 52 ∗ 64	26 ∗ 26 ∗ 128	3	199360512	2.01 ms	198.37
26 ∗ 26 ∗ 128	13 ∗ 13 ∗ 256	3	199360512	2.43 ms	164.08
13 ∗ 13 ∗ 256	13 ∗ 13 ∗ 512	3	199360512	4.07 ms	97.97
13 ∗ 13 ∗ 512	13 ∗ 13 ∗ 1024	3	797442048	16.28 ms	97.97
13 ∗ 13 ∗ 1024	13 ∗ 13 ∗ 256	1	44302336	4.67 ms	18.97
13 ∗ 13 ∗ 256	13 ∗ 13 ∗ 512	3	199360512	4.07 ms	97.97
13 ∗ 13 ∗ 512	13 ∗ 13 ∗ 256	1	22151168	2.34 ms	18.93
13 ∗ 13 ∗ 256	13 ∗ 13 ∗ 128	1	5537792	0.58 ms	19.10
13 ∗ 13 ∗ 384	13 ∗ 13 ∗ 256	3	149520384	3.05 ms	98.05
13 ∗ 13 ∗ 256	13 ∗ 13 ∗ 256	1	11075584	1.2 ms	18.46
Total			2300980176	46.95	98.01

perform linear estimation of performance parameters). Then, we divided these works into four parts: instruction, on-chip buffer, computing architecture, and memory access to discuss the estimation, showing the reader a relatively quantitative module evaluation. Finally, we analyze the calculation density, calculation energy efficiency, and generality of the acceleration system to contrast with the work of this paper.

The specific comparison results are shown in Table 6. We compare the modules added in this article with selected papers. Among them, the Ref. [19] is a customized accelerator system, which does not have an instruction system, on-chip cache, and memory management system. At the same time, the computing architecture also uses a different systolic array from this paper. Therefore, the calculated density and energy efficiency ratio is lower than this article. Reference [28] added the instruction subsystem, and the computing architecture is also in the form of full pipeline, so its computing density is similar to that of this paper, and it is universal. But it does not have an on-chip cache subsystem, so its energy efficiency ratio is far lower than this article. Reference [14] has an instruction subsystem, an on-chip cache subsystem, and a pipeline computing architecture. However, because it is based on HLS for modeling, the optimization of its full-pipeline computing architecture is not as good as this article and literature [28], and the architecture design of its on-chip cache subsystem is worse than this article. Therefore, the calculation density is slightly lower than that of this paper and literature [28], and the energy efficiency is higher than that of the literature without an on-chip cache subsystem [28]. The MMU subsystem added in this article is a customization of the general-purpose DMA. On the one hand, it improves the operating efficiency of the memory access system and makes a certain contribution to the calculation

Table 6: Comparison between different accelerations.

Model	Inst.	TPOCM	Architecture	MMU	GOPS/DSP	GOPS/power	Generality
[19]	×	×	×	×	0.081	13.84	×
[28]	√	×	√	×	0.13	6.22	√
[14]	√	√	√	×	0.112	15.04	√
Full	√	√	√	√	0.14	21.68	√

Table 7: Comparison between different accelerations.

Parameter	[29]	[28]	[30]	[31]	[14]	This work
Platform	Virtex-7	Zedboard	Z7045	Z7045	UltraScale+	Z-7100
BRAM	89	185	486	486	491	155
DSP	565	160	780	771	609	673
Frequency (MHz)	150	-	150	200	300	210
Precision	-	16 bits	16 bits	16 bits	16 bits	16 bits
Power	-	3.36	9.63	9.3	11.8	4.52
Performance (GOP/s)	20.62	10.45	187.8	62.54	102	98.01 (peak 198.37/layer)
GOPS/DSP	0.036	0.07	0.24	0.081	0.16	0.14
GOPS/W	-	3.11	19.50	6.72	8.64	21.68

density; on the other hand, it removes the partial redundancy of the general DMA, which contributes to the reduction of the on-chip area and saves on-chip resources.

5.4. Discussion. Table 7 summarizes the key performance parameters of different types of accelerators and compares them with the work in this paper. Among them, the performance of [28, 29] is much lower than that of ours. On the one hand, there are reasons for the limitation of the number of DSP hard cores. On the other hand, the clock frequency also has a certain impact. But paying attention to the index of computing energy efficiency and computing density, it can be seen that [28, 29] are lower than ours to some extent. Therefore, the accelerator in this paper is superior to [28, 29] both in critical path optimization and in the control of computational performance and energy consumption. Focusing on the performance parameters of [30], its performance has been greatly improved, and it is better than ours in terms of computational density. There are two main reasons for this. On the one hand, it uses more hard-core DSP for calculations than ours. On the other hand, it uses several times on-chip storage resources than ours, so it has enough on-chip space to store computing data, which can greatly reduce access to off-chip storage and reduce data access time. But doing so will also greatly increase the system power consumption, so its computational energy efficiency is lower than the accelerator in this paper. Looking at [31] again, in terms of the on-chip storage and computing resource consumption, it surpasses ours, but its computing performance is lower than ours, and its power consumption exceeds ours. This is because its core utilization is way low. The computing performance of [14] is almost the same as that of ours, but it uses UltraScale devices, which have better process and can achieve higher computing frequencies. However, [20] uses HLS for modeling, which may lead to its optimization of computational density not as good as that of RTL level modeling in this paper. So overall, the optimization of power consumption in this paper is mainly achieved by reducing the use of on-chip storage and establishing an independent memory management system. In this paper, the critical path is optimized by reducing the logic delay to control the critical path to the length of a DSP delay. However, when the static command control computing architecture was expanded, it was found that the net delay was too large, so the net delay was further shortened by reducing the fan-out of the single data bus.

6. Conclusion and Future Research

This paper presents a set of workflows for a deep convolutional neural network accelerator with a certain generality. First of all, we propose a set of basic operation instructions based on the calculation characteristics of the classic convolutional neural network at this stage. These instructions can guide the accelerator system to scale the architecture before synthesis and dynamic reconfiguration after synthesis. Then, this paper proposes a fully pipelined parallel computing architecture based on the instruction set. Compared with the current accelerator based on the reconfigurable platform, it has the characteristics of deep scalability and reconfiguration, low resource consumption, and low energy consumption. These features allow our accelerators to not only accelerate algorithms on resource- and energy-constrained reconfigurable platforms but also perform high-performance real-time calculations on resource-rich reconfigurable platforms. It not only can provide calculations for a specific algorithm but also can provide computing requirements for convolutional neural

networks with different algorithm structures through dynamic instruction reconfiguration. This also means that the accelerator designed in this paper can make a dynamic balance between inference speed, resource usage, and power consumption.

Finally, we test the accelerator performance on YOLOv2-tiny and yolov3-tiny. Under the 4IN-16OUT accelerator architecture, they reached 98.01 GOPs and the peak value reaches 198.37 GOPs. Among them, YOLOv3-tiny takes 46.95 ms to process a picture with an input size of 416 ∗ 416 and achieves processing performance of 21 FPS. The acceleration core is finally implemented on XC7Z100, using 16-bit quantization bit width, which takes up 36% of DSP resources, 28% of LUT resources, and 21% of BRAM resources. Finally, compared with the existing equivalent accelerator design flow, the energy efficiency ratio of 21.68 GOP/W and the calculation density of 0.14 GOP/DSP of the accelerator in this paper are better than the current accelerator to some extent. Although the performance will be lower than some accelerators, the accelerator in this paper greatly improves the flexibility of the accelerator while meeting the performance requirements of real-time applications.

Currently, we are working on the specific application of the accelerator system. We customize a highly integrated heterogeneous platform for it so that it can be directly mounted on our existing deep-sea cameras for real-time edge computing. The ultimate goal is to apply it to underwater equipment, such as underwater landers and AUVs, to meet the intelligent tasks of underwater equipment. Subsequently, we will also make the following further optimization work on the acceleration system itself. First, the parameters and calculations will be further quantified, so that the parameters and calculations can run in a lower bandwidth, which can greatly save on-chip storage resources and improve computing performance. Second, the convolution array utilization will be further optimized, specifically to improve the data continuity and multiplexing rate on the chip. Thirdly, the computing mode of the accelerator computing core will be optimized to adapt to more types of algorithm structures to further improve the universality of the accelerator.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Key Research and Development Plan of China (Y820043001): Research on deep-sea video acquisition, transmission, and processing technology, system integration and demonstration applications (Research on deep-sea wireless optical communication and long-distance optical fiber micro cable communication technology).

References

[1] H. Kim, H. Nam, W. Jung, and J. Lee, "Performance analysis of CNN frameworks for GPUs," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Santa Rosa, California, April 2017.

[2] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of GPU-based convolutional neural networks," in *2010 18th Euromicro conference on parallel, distributed and network-based processing*, Pisa, Italy, 2010.

[3] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: algorithm-soc co-design for low-power mobile continuous vision," 2018, https://arxiv.org/abs/1803.11232.

[4] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs," in *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, Atlanta, GA, USA, 2016.

[5] J. Caba, M. Díaz, J. Barba, R. Guerra, and J. A. T. S. López, "Fpga-based on-board hyperspectral imaging compression: benchmarking performance and energy efficiency against gpu implementations," *Remote Sensing*, vol. 12, no. 22, p. 3741, 2020.

[6] J. M. Haut, S. Bernabe, M. E. Paoletti, R. Fernandez-Beltran, A. Plaza, and J. Plaza, "Low–high-power consumption architectures for deep-learning models applied to hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 5, pp. 776–780, 2019.

[7] M. Teichmann, M. Weber, M. Zollner, R. Cipolla, and R. Urtasun, "Multinet: real-time joint semantic reasoning for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Suzhou, China, June 2018.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, https://arxiv.org/abs/1510.00149.

[9] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: towards lossless cnns with low-precision weights," 2017, https://arxiv.org/abs/1702.03044.

[10] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, "Post-training 4-bit quantization of convolution networks for rapid-deployment," 2018, https://arxiv.org/abs/1810.05723.

[11] S. Lin, R. Ji, C. Yan et al., "Towards optimal structured cnn pruning via generative adversarial learning," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, USA, 2019.

[12] Qin and Cao, "Design of convolutional neural network hardware accelerator based on FPGA," *Journal of Electronics and Information*, vol. 41, no. 11, pp. 2599–2605, 2019.

[13] M. Sankaradas, V. Jakkula, S. Cadambi et al., "A massively parallel coprocessor for convolutional neural networks," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, Boston, MA, USA, July 2009.

[14] S. Zhang, J. Cao, Q. Zhang, Q. Zhang, Y. Zhang, and Y. Wang, "An fpga-based reconfigurable cnn accelerator for yolo," in

2020 IEEE 3rd International Conference on Electronics Technology (ICET), Chengdu, China, May 2020.

[15] S. Hareth, H. Mostafa, and K. A. Shehata, "Low power CNN hardware FPGA implementation," in *2019 31st International Conference on Microelectronics (ICM)*, Cairo, Egypt, 2019.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, June 2016.

[17] D. Hong, J. Yao, D. Meng, Z. Xu, and J. Chanussot, "Multimodal GANs: toward crossmodal hyperspectral–multispectral image segmentation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 6, pp. 5103–5113, 2020.

[18] D. Hong, L. Gao, N. Yokoya et al., "More diverse means better: multimodal deep learning meets remote-sensing imagery classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 5, pp. 4340–4354, 2020.

[19] D. Hong, L. Gao, J. Yao, B. Zhang, A. Plaza, and J. Chanussot, "Graph convolutional networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 7, pp. 5966–5978, 2020.

[20] L. Chen, Z. Jiang, L. Tong et al., "Perceptual underwater image enhancement with deep learning and physical priors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 8, pp. 3078–3092, 2020.

[21] H. Ma, F. Mao, and G. W. Taylor, *Theano-mpi: A Theano-Based Distributed Training Framework*, Springer, 2016.

[22] T. Chen, Z. du, N. Sun et al., "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[23] Y. Chen, T. Luo, S. Liu et al., "Dadiannao: a machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Cambridge, United Kingdom, December 2014.

[24] Z. Du, R. Fasthuber, T. Chen et al., "ShiDianNao: shifting vision processing closer to the sensor," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, Portland Oregon, June 2015.

[25] D. Liu, T. Chen, S. Liu et al., "PuDianNao: a polyvalent machine learning accelerator," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 369–381, 2015.

[26] P. Meloni, G. Deriu, F. Conti, I. Loi, L. Raffo, and L. Benini, "A high-efficiency runtime reconfigurable IP for CNN acceleration on a mid-range all-programmable SoC," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, November 2016.

[27] C. Zhang, J. S. Di Wu, G. Sun, G. Luo, and J. Cong, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, San Francisco, USA, August 2016.

[28] Z. Yu and C.-S. Bouganis, "A parameterisable FPGA-tailored architecture for YOLOv3-tiny," in *International Symposium on Applied Reconfigurable Computing*, Springer, 2020.

[29] W. Wei, Z. Kai-li, W. Yi-chang, W. Guang, Y. Zheng-lin, and Y. Jun, "FPGA parallel structure design of convolutional neural network (CNN) algorithm," *Microellectronics and Computer*, vol. 36, no. 4, pp. 57–62, 2019.

[30] K. Guo, L. Sui, J. Qiu et al., "Angel-eye: a complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2017.

[31] C. Qiu, X.'. Wang, T. Zhao, Q. Li, B. Wang, and H. Wang, "An FPGA-based convolutional neural network coprocessor," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 3768724, 12 pages, 2021.