Agile Design and Implementation of a Systolic Array-Based CNN Accelerator

Hongyu Wei

Abstract— **Since its birth, convolutional neural network (CNN) has been widely used in many fields. FPGA implementations of CNNs have attracted widespread attention due to their high performance and energy efficiency. However, some current computing architectures do not fully utilize the computing power of FPGAs and can only accelerate a single network. In addition, the traditional method of developing FPGA accelerators using Verilog cannot meet the diverse needs and flexibility of accelerators. Therefore, this paper proposes a parameterized configuration of a general convolutional neural network accelerator. In order to improve the computing throughput and frequency, we adopt a systolic array architecture to implement the computing unit of this accelerator. Furthermore, in order to effectively meet the diverse needs of industry and academia, we adopted an agile development approach using Spi-nalHDL. The accelerator was ultimately deployed on various boards such as VU9P and tested using representative algorithms in convolutional neural networks (YOLOv4-Tiny). Experimental results show that when the accelerator runs at a frequency of 200 MHz and accelerates the YOLOv4-Tiny algorithm, the FPS is 85.09. Has excellent acceleration effect.**

Index Terms—**FPGA, Systolic array, CNN accelerator.**

I. Introduction

Research on FPGA-based convolutional neural network acceleration[1] has become a focal point in the fields of deep learning and computer vision. FPGAs enable customized designs tailored to different application scenarios and requirements. They can accelerate convolutional operations through techniques such as parallelization and pipelining, significantly enhancing the computational efficiency of CNNs. Moreover, FPGAs support dynamic adjustment and real-time deployment of networks, making them highly promising for low-power scenarios like embedded systems and mobile devices.

In current research, FPGA-based acceleration methods for CNNs can be categorized into two main approaches: dataflow-based methods and architecture-based methods. Dataflow-based methods decompose convolution operations into sub-operations and execute them in parallel using pipelining on FPGA to improve computational efficiency. Architecture-based methods optimize FPGA architectures, such as incorporating on-chip memory and utilizing multi-level caching, to reduce storage access and data transfer overhead and improve computational efficiency. In our design, we combine these two approaches by employing a systolic array structure for fine-grained decomposition and parallel computation of convolution operations. Additionally, we make efficient use of on-chip memory on FPGA boards for data storage and reuse, achieving high-performance parallel computation while reducing power consumption.

To address the current phenomenon where neural network accelerators are developed based on specific networks, we have employed an agile development approach based on Spinal HDL. This approach enables a more versatile and adaptable accelerator design capable of accommodating different networks. Using Spinal HDL for agile implementation of the neural network accelerator, we have developed different operators as plugins. By modifying the corresponding parameters, dedicated neural network accelerators can be generated for different convolutional neural networks. By embracing the principles of agile development and hardware-software co-design, we have successfully completed this task and achieved remarkable acceleration results.

Manuscript received March 18, 2024

Hongyu Wei, School of computer science and technology, Tiangong University, Tianjin, China

II. Related Work

A. Convolutional Neural Networks

In 1994, Yann LeCun[2] proposed the Convolutional Neural Network (CNN) . As a typical algorithm in deep neural networks[3], CNN is used in video and image data processing with multi-layer network structures. CNN shines in the field of computer vision and can efficiently solve tasks such as target detection[4], image segmentation[5], and image classification[6]. Since the convolutional neural network contains a convolutional layer (Convolutional Layer) and a pooling layer (Pooling Layer), the CNN can quickly and effectively extract local features of the data while reducing the amount of model parameters, making the neural network model more efficient. The complexity is further reduced and the generalization ability of the model is improved.

Most CNNs are composed of convolutional layers, pooling layers, fully connected layers, pooling functions, etc. The most critical one is the convolution layer, which contains multiple convolution kernels. The convolution kernels are also called filters. In the convolutional layer, the convolution kernel performs a sliding window convolution operation on the input image to extract the feature information of the input image. The result of the convolution operation is a new feature map (Feature Map), and each value in the feature map represents the feature response at that location. Generally speaking, the convolutional layer will be followed by a nonlinear activation function, such as ReLU (Rectified Linear

Unit), etc., with the purpose of introducing nonlinear transformation into the model to enhance the expression ability of the model. In neural networks, the purpose of the pooling layer is to reduce the spatial dimension of the feature map, so that the main features can be preserved while reducing the computational complexity of the model. Max Pooling and Average Pooling are two common operations in the pooling layer. The fully connected layer often appears after multiple convolution operations and pooling operations, and is mainly used to summarize and output the information of the feature map to the final classification or regression layer.

B. *Agile design of accelerator based on Spinal HDL*

Spinal HDL is a modern hardware description language, based on the Scala language, designed to bring a higher level of abstraction and flexibility to digital circuit design. Its developer is Cocotb engineer Charles Papon, and the first version was released in 2017. The design goal of Spinal HDL is to combine the advantages of hardware description languages and modern programming languages to provide a more intuitive and efficient way to describe and design digital circuits.

Spinal HDL language, as an extension library of Scala language, allows designers to use the features of Scala language to improve the maintainability of circuits and improve development efficiency when designing circuits. We can understand Spinal HDL as an efficient Verilog generator. After the designer uses the Spinal HDL language to design the circuit code, it will be compiled into a source program based on the Scala language, and then the final Verilog code will be generated through the Spinal compiler. Therefore, all third-party tool libraries and development processes on the market that support Verilog can be seamlessly connected to the Spinal HDL language. One of the characteristics of Spinal HDL's high-efficiency development is that it integrates automatic logic checks to automatically correct and prompt problems such as port direction errors, bit width mismatches, signal drive loops, and cross-clock domains. It is effective It reduces the development complexity and shortens the circuit development cycle. At the same time, circuit codes developed using Spinal HDL are highly readable and easy to maintain, making it easier for designers to reuse chip codes.

In the field of digital circuits, design and simulation are inseparable, and the case construction method of using the high-level language Scala has obvious advantages. The language also integrates an API that provides functions for reading and writing DUT signals, branching and joining the simulation process, sleeping and waiting for given conditions to be reached. In this way, we can easily combine the test platform with common Scala unit testing frameworks, supporting not only SpinalSim but also ScalaTest. In addition, Spinal HDL also integrates external simulators, including Verilator, GHDL, Icarus Verilog, etc., and is also compatible with VCS, XSIM in Vivado, etc. for simulation. The simulation process using Spinal HDL is shown in Figure 1. To sum up, this article uses Spinal HDL to perform agile design of deep learning accelerator.

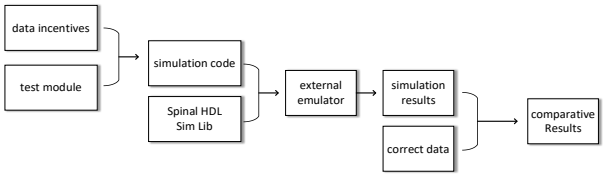

Figure 1. Spinal HDL simulation flow chart

C. *Systolic array*

Systolic array is a parallel computing architecture that consists of many processing units distributed in a grid-like structure. Each processing unit has its own local memory and control logic and is able to perform computing tasks independently. Systolic array architecture was first used in early parallel computer systems, such as array processors and data flow computers, to accelerate tasks such as numerical calculations and signal processing.

The systolic array was first proposed by American computer scientist H.T. Kung in 1982[7]. Early systolic array computer systems typically consisted of multiple processing units that communicated and collaborated through an interconnection network. Each processing unit can perform computing tasks independently and can exchange data and coordinate processing with other processing units. Systolic array architecture is designed to achieve a high degree of parallelism and flexibility to improve the overall performance and throughput of computer systems. With the continuous development of computer technology, systolic array architecture has gradually evolved into various forms, including multi-core processors, GPUs (graphics processing units) and FPGAs (field programmable gate arrays). These new systolic array architectures have been widely used in different fields, such as scientific computing, image processing, signal processing, and artificial intelligence. Figure 2 shows a common two-dimensional systolic array structure.

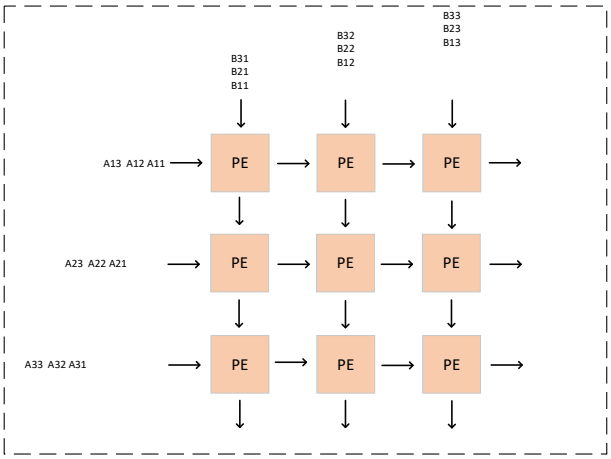

Figure 2. Two-dimensional systolic array structure

III. ACCELERATOR DESIGN

A. *Accelerator architecture design*

The accelerator architecture design is shown in Figure 3. HSOT is the CPU, which is mainly responsible for the three tasks of sending data, post-processing the model, and visualizing the processing results. By using the IP core of

Xilinx XDMA, PCIe is controlled to realize data interaction between PS and PL and the scheduling of execution tasks. In terms of the overall process, the PS side first sends the image data and weight data to the DDR through the AXI bus using PCIe. And the instruction set is transmitted to the on-chip cache through the AXI-Lite bus using PCIe. This is because the data volume of the picture and weight data is large, and it is suitable for the complete AXI protocol with bursts. The instruction set is a small batch of data. If it is also transmitted using the complete AXI protocol, it will cause a certain waste of resources. Therefore, the AXI-Lite protocol that cannot be transmitted in bursts is used for transmission. The analysis of the instruction set is completed through the Instruction module. Through the analysis of the instructions, various parameters required for the current network layer calculation will be obtained, and these parameters will be transferred to the Global Controller, Data Pre-processing, ALU and other modules. These parameters include but are not limited to the calculation type of the current network layer, the size of the image and weights, the number of channels and the storage location of the data in the DDR, the parameters required for quantization, etc. After obtaining the storage location parameters of the image and weight, the DMA Group transfers the data from the DDR to the Data Pre-processing module through the AXI bus for preprocessing. This module will process the incoming image data so that the data format meets the operation requirements of the ALU module. After data pre-processing, Data Pre-processing will send the data to the ALU module through AXI-Stream. At the same time, the parameters parsed by the Instrucion module will also be sent to the ALU module. The ALU module will choose to call the Systolic Array for convolution calculation or the Reshape module through SWITCH based on the parameters parsed in the instruction. The calculation results of the ALU module will also be sent to the Buffer Group for buffering through AXI-Stream. After the Quantization Unit, that is, the quantization module, is ready, it is sent to the quantization module in the form of AXI-Stream. After the data is processed by the quantization module, the DMA Group sends the final result to the specified location in the DDR according to the parameters of the storage location. After the neural network is completely executed, the final calculation results in the DDR are returned to the PS side through PCIe for post-processing and other operations.

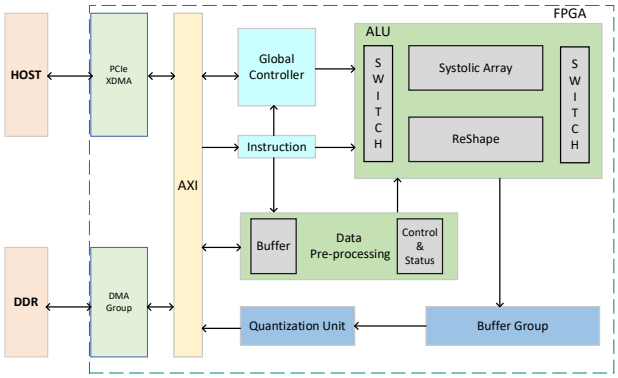

Figure 3. Accelerator architecture

B. *ALU architecture design*

The ALU module is the core computing module of the accelerator. It contains not only the systolic array computing module but also the ReShape module. It can be said that the performance of the ALU determines the overall performance of the accelerator.

The overall architecture of the ALU is shown in Figure 4. The input of the ALU module is multiple sets of AXI-Stream streams, which corresponds to the multiple sets of DMA structures we designed in Block Design. Multiple sets of AXI-Stream streams will input multiple input feature maps and weight data into the ALU module. Multiple groups of data received will first enter the Buffer Group for buffering. Since FPGA resources are limited, it is impossible to put all input feature maps and weight data into the Buffer. Therefore, we adopted the form of line cache in the design and adopted channel priority. At the same time, the ALU module will also receive the instruction parameters parsed by the Instruction module during calculation, and can determine from the parameters whether the current network layer enables the systolic array module for convolution calculation or the ReShape module. When convolution calculation is performed, the feature map data and weight data will flow into the Systolic Array through SWITCH. After the Systolic Array calculation is completed, the data will flow into the Buffer through SWITCH, and finally flow out of the ALU module in the form of AXI-Stream stream. If the incoming command parameter chooses to enable ReShape, multiple feature map data will enter the ReShape module in the form of AXI-Stream stream for calculation. The calculation results will also flow into the Buffer through SWITCH, and finally flow out of the ALU in the form of AXI-Stream stream. module, passed to downstream.

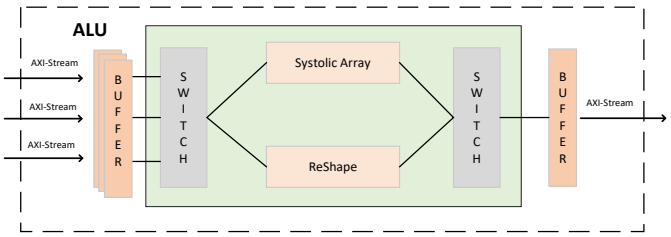

Figure 4. ALU architecture

C. *Three-dimensional systolic array structure*

Using systolic arrays for calculations will bring huge computational gains. In order to cater to the large data volume and high parallelism characteristics of convolution calculation, our accelerator adopts a three-dimensional systolic array structure with customized size, as shown in Figure 5. Each PE (processing unit) moves the weights and feature maps to adjacent PEs after calculation in each cycle, keeping them in a flowing state. For the entire systolic array result, buffering is applied not only to the IB Buffer (input feature map cache) and WB Buffer (weight cache) in the array, but also to the inside of each PE. All input feature map data and weights will flow into the IB Buffer and WB Buffer, then flow into the PE, and then interconnect with adjacent PEs through the PE's internal Buffer. There are similar results in OB (output buffer) to achieve seamless pipeline connection.

Since the layout of the array is similar to the arrangement of the underlying resources of the FPGA, it is more conducive to timing convergence and reduces routing complexity. Thanks to the support of Spinal HDL language, the size of the entire array, the number of PEs, the size of the Buffer, etc. are all parameterized and customized. The entire array can be customized in detail to meet specific needs. When the FPGA board resources are limited, an array structure with smaller size and buffer can be generated. For application scenarios with high real-time performance, a larger array structure can be generated to meet the demand.

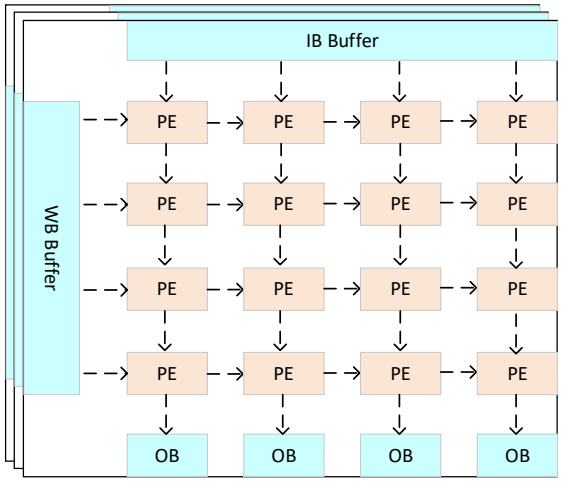

Figure 5. Three-dimensional systolic array

D. *Mapping of convolution calculations*

Convolutional calculations occupy a major position in neural networks. The process of 3*3 size convolution calculation is shown in Figure 6. The corresponding feature window of the input feature performs multiplication and addition operations with each convolution kernel to form an output feature map. Each convolution kernel corresponds to each channel of a single pixel in the output feature map.

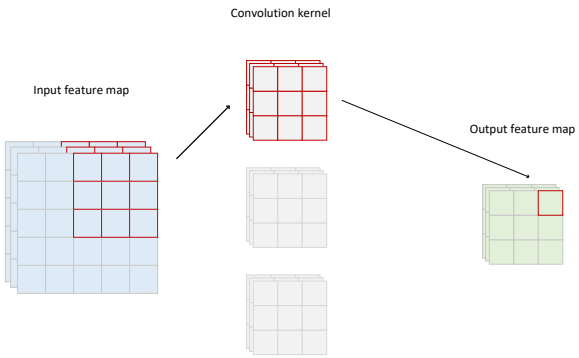

Figure 6. 3*3 convolution calculation example

Convolution calculations can be split in different dimensions. In this design, the convolution calculation is split differently from channels, images, and convolution kernels to map to different dimensions of the systolic array, and data reuse is performed to varying degrees, as shown in Figure 7 . In this design, the input feature window flows in from the top of the array, the convolution kernel flows in from the left side of the array, and the convolution size is mapped to the third dimension. If it is a 2x2 convolution, the number of the third dimension is 4, corresponding to Calculation of 4 pixels. The feature map information, weights, and current PE calculation results in each clock cycle PE will continue to flow to adjacent PEs to complete the corresponding operations. The data flowing into the input feature map and convolution kernel is channel-first, that is, all channel information will be processed first before the new sliding window is processed. Specifically, if it is a 3x3 array, the three channel information of the first input feature map sliding window will flow into the top of the array in sequence, realizing parallel operations of the input channels. The information of the convolution kernel will flow into the left side of the array in sequence. In the flow of the convolution kernel, the channel information flows in sequence. Only the data of the channel corresponding to the feature map will be calculated, thus realizing parallel operations between the output channels. However, the result of such calculation is only incomplete information of different channels and different convolution kernels. After flowing out of the pulsation array, these incomplete information will be accumulated in the corresponding channel and convolution size direction, and finally a complete output will be obtained. Channel results.

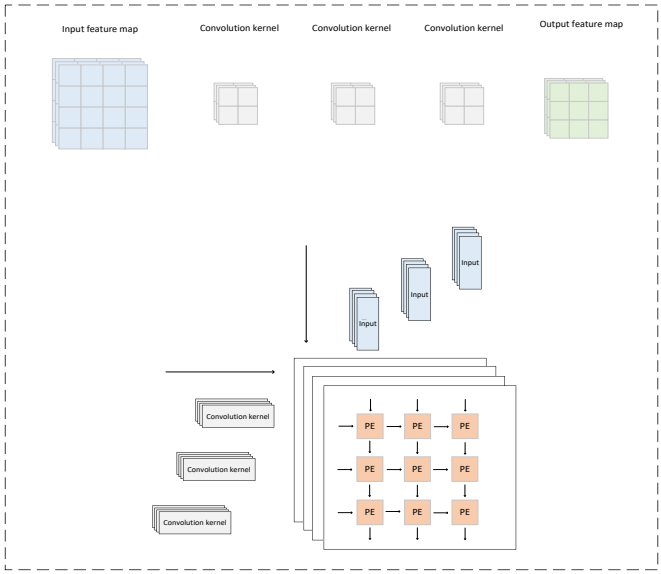

Figure 7. Convolution mapping example

E. *PE module design*

The PE (Processing Element) unit in the systolic array structure refers to the basic computing unit that performs specific computing tasks. PE units typically perform various mathematical operations and signal processing operations and contain storage units for temporary storage of input data, intermediate results, and output data. These storage units may be registers, memory units or caches. Data transmission and exchange are performed between PE units through communication interfaces. These communication interfaces usually include input ports and output ports, which are used to receive input data and transfer calculation results to other PE units. PE units are usually organized together in the form of arrays to form a large-scale parallel computing structure.

Generally speaking, the PE unit is composed of DSP. In order to better improve the operating frequency of the system, align the input data, have higher controllability, and the ability

to interrupt and respond to back pressure at any time, we designed a controllable buffer with The PE operation unit of the structure is shown in Figure 8.

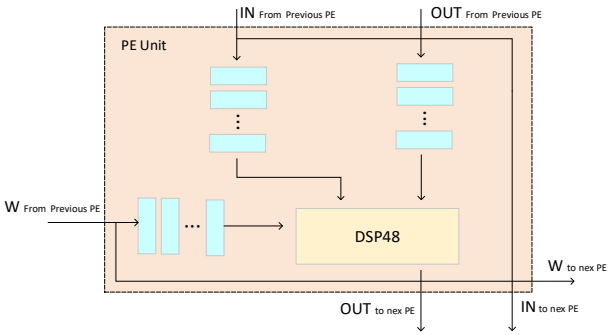

Figure 8. PE architecture diagram

Each PE computing unit has three inputs and three outputs. The three inputs are the feature map (IN), weight (W) and the calculation result of the previous PE (OUT) passed in from the adjacent PE. The three output ports transmit the incoming feature maps, weights and calculation results of this module to the next PE respectively. After getting the input data, if the downstream module is allowed to accept the data, the feature map data and weight data in the buffer will be transferred to the DSP for multiplication and addition operations. The calculation result of the previous module is the feature map and different convolution kernels. The calculation result cannot be added to the current result. It needs to wait for the current result to be calculated and transferred to the adjacent PE together.

IV. EXPERIMENT

A. *Experimental environment*

Different FPGA platforms were used for testing in this experiment, in order to test the versatility of the accelerator on different resource platforms. In the hardware platform, a heterogeneous acceleration solution of CPU+FPGA is used to accelerate the neural network. The CPU uses Intel Core i7-11800H@2.30GHz, the memory is 16GB, the operating system is Ubantu21.04, and the FPGA uses Xilinx VU9P, 325T and ZYNQ7100.

The development board with the VU9P chip in the experiment is the FX609QL development version of Feishu Technology. This development board supports PCIe4.0 X8 mode transmission method and has 16GB DDR4 and 1Gbit FLASH. In the experiment, we used Xilinx's XDMA host computer driver to interact with the CPU through PCIe.

The development board with 325T chip is Milianke MK7325FA development board. The board has 4 pieces of 512Mbyte, a total of 2GB of DDR3 storage capacity, 256Mbit FLASH, supports PCIe and Gigabit Ethernet for data transmission, and has SATA, USB, HDMI and other interfaces. In the experiment, the XDMA driver was also used to interact with the CPU-side data through PCIe.

The ZYNQ series 7100 development board uses the Milianke MZ7100FA development board. The development board has a Cortex-A9 dual-frequency main core, PS side and PL side each have 1GB of DDR3 storage capacity, and has multiple interfaces such as PCIe, SATA, and Gigabit Ethernet. The Ethernet interface on the PS side is used for data exchange on this board.

This experiment performed inference acceleration on YOLOv4-Tiny to verify the flexibility and versatility of the accelerator developed by Spinal HDL language. The data set uses VOC2007, which consists of aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor, etc. 20 Each category is composed of training set, verification set and test set according to the ratio of 8:1:1. The image size is 416×416. The quantification is completed with the help of Pytorch1.7.

B. *Analysis of results*

Table 1 shows the resource consumption of the accelerator under a single-core architecture. Channel In and Channel Out represent the computing parallelism of the input channel and the output channel, which is reflected in the size of the array in the systolic array. Since the BRAM of 325T and 7100 only has 445 and 755 blocks, and the DSP has only 840 and 2020 blocks, it is not enough to support increasing the parallelism of the accelerator, so the parallelism is set to 8 for experiments. The VU9P not only has 2160 BRAMs, but also has 960 URAM resources unique to the board. The DSP resources have reached 6840, which is very rich in resources. Therefore, we conducted 3 sets of parallelism experiments on the VU9P board, which were 4, 8, and 16 respectively. During the synthesis process, you can set whether to use URAM. We chose to use URAM in the experiment. It is easy to see from the figure that as the degree of parallelism increases, the usage of BRAM and URAM will also increase. The number of DSPs used in the core systolic array module in the experiment is also closely related to the degree of parallelism. When the size of the systolic array becomes larger, the degree of parallelism increases and the number of DSPs used will also increase.

Table1. Resource consumption

	325T	7100	VU9P	VU9P	VU9P
Channel In	8	8	4	8	16
Channel Out	8	8	4	8	16
LUT	70536	44307	60289	77806	136806
BRAM	266.5	243	120	164	244.5
URAM	0	0	37	73	290
DSP	474	474	173	477	1525
Power(W)	11.803	9.042	8.448	10.314	15.136

Table 2 shows the acceleration effect of YOLOv4-Tiny at 200MHz frequency. It can be seen that with the improvement of computing parallelism, FPS has been significantly improved. When accelerating the YOLOv4-Tiny algorithm, when the parallelism is set to 16 on VU9P, the FPS improvement is greatly improved compared to when the parallelism is 4. When the calculation parallelism is 8, the acceleration effects of the two algorithms on 325T, 7100, and VU9P are not exactly the same. This is because the DDR and other hardware configurations of these three development boards are different, and the interface speed The differences lead to different acceleration effects.

Table2. Accelerate performance

	325T	7100	VU9P	VU9P	VU9P
Channel In	8	8	4	8	16
Channel Out	8	8	4	8	16
FPS	20.83	22.16	4.63	21.53	85.09
Power(W)	11.803	9.042	8.448	10.314	15.136

V. CONCLUSION

This paper designs a convolutional neural network accelerator based on the systolic array structure and implements it quickly through the Spinal HDL language, so that the accelerator can be easily adapted to a variety of deep neural networks and integrate the operators in the network in the form of plug-ins. in the accelerator. And based on the language features of Spinal HDL, the calculation granularity of the accelerator can be adjusted to adapt to different scene requirements. Using a three-dimensional systolic array, a regularly routed structure, to perform core calculations allows the accelerator to work at higher frequencies and have higher computing performance. Finally, we took YOLOv4-Tiny in the convolutional neural network as an example to conduct multi-dimensional tests on the accelerator, conducting experiments from different FPGA development boards and different computing parallelism. Experimental results show that the accelerator in this paper not only has good performance, but also takes into account versatility and flexibility, and is suitable for a wide range of application scenarios.

REFERENCES

[1] Mittal S. A survey of FPGA-based accelerators for convolutional neural networks[J]. Neural computing and applications, 2020, 32(4): 1109-1139.

[2] LeCun Y, Bengio Y. Convolutional networks for images, speech, and time series[J]. The handbook of brain theory and neural networks, 1995, 3361(10): 1995.

[3] Canziani A, Paszke A, Culurciello E. An analysis of deep neural network models for practical applications[J]. arXiv preprint arXiv:1605.07678, 2016.

[4] Yan A, Li J, Sun B, et al. Research on moving target tracking system based on FPGA[C]//2020 Chinese Control And Decision Conference (CCDC). IEEE, 2020: 1667-1671.

[5] Liu Y, Wang Y, Chang L, et al. A fast and efficient FPGA-based level set hardware accelerator for image segmentation[C]//2020 IEEE international conference on integrated circuits, technologies and applications (ICTA). IEEE, 2020: 61-62.

[6] Saidi A, Othman S B, Dhouibi M, et al. FPGA-based implementation of classification techniques: A survey[J]. Integration, 2021, 81: 280-299.

[7] Kung H T. Why systolic architectures?[J]. Computer, 1982, 15(1): 37-46.