Transactions on Computational and Scientific Methods | Vo. 4, No. 9, 2024
ISSN: 2998-8780
https://pspress.org/index.php/tcsm
Pinnacle Science Press

Dynamic Reconfigurable CNN Accelerator for Embedded Edge Computing: A Hardware-Software Co-Design Approach to Minimize Power and Resource Consumption

Carson Hale
Department of Electrical and Computer Engineering, University of Denver, Denver, USA
carsonr86@du.edu

Abstract: With the growing demand for efficient Convolutional Neural Network (CNN) deployments in low-power, edge computing environments, FPGAs have emerged as an ideal platform due to their parallel processing, low power consumption, and dynamic reconfiguration capabilities. This paper presents the design and implementation of a dynamic reconfigurable CNN accelerator tailored for embedded edge devices. The proposed accelerator architecture is designed with hardware-software co-design principles, optimizing for modularity and flexibility. Experimental results demonstrate significant reductions in power consumption (42.06%) and hardware resource utilization (17.76% for FF, 32.82% for LUT, 48.70% for BRAM, and 47.01% for DSP) when compared to static circuit accelerators. These improvements highlight the potential of the accelerator for a broad range of CNN applications, particularly in resource-constrained environments. Future work will focus on further optimization and extending the architecture to support CNN-based lightweight object detection algorithms.

Keywords: FPGA; CCN; Dynamic Reconfiguration; Hardware Acceleration; Software Hardware Co-design.

1. Introduction

With the continued development of Convolutional Neural Network (CNN) algorithms and the improvement of FPGA computational performance, FPGA-based CNN accelerators are becoming increasingly popular in industrial and defense AI edge applications [1]. They are widely used in low-power embedded image classification [2], object detection [3], and biometric recognition [4], among others. As CNN algorithms are computationally intensive, they have large algorithmic complexity and a number of parameters when constrained to high accuracy. When deployed on edge embedded computing devices that are limited in power consumption and resource volume, Application Specific Integrated Circuits (ASICs) and Graphics Processing Units (GPUs) struggle to meet the requirements of low power consumption, use of fewer hardware resource area and real-time performance [5]. On the other hand, FPGAs with high parallel computing [6], low power consumption [7] and dynamic reconfiguration [8] capabilities are well suited for device requirements.

In recent years, several CNN accelerator designs have been proposed. Literature [9] proposed a three-layer storage system CNN accelerator that uses channel interleaving, multi-channel transmission, and multi-level co-optimization methods. However, excessive channel interleaving causes long data delays and results in hardware resource redundancy. Literature [10] proposed a general-purpose CNN accelerator for System on a Chip (SoC) design, which supports any input size, input feature map depth, and stride by encapsulating the entire CNN network as an operation accelerator core. The accelerator uses the main memory to store data between computation layers, splitting complex convolutions into sub-convolutions to improve generality, but its accelerator inference speed is only 0.83 FPS. Literature [11] proposed a CNN accelerator that uses loop tiling and data flow modeling optimization, improving the acceleration effect while reducing hardware resource utilization.

The paper proposes a low-power, high-flexibility, and hardware resource space-reusable dynamic reconfigurable Convolutional Neural Network (CNN) accelerator for embedded edge computing devices. The main research contents are summarized as: (1) a dynamic reconfigurable CNN accelerator architecture is proposed; (2) the operation modules are designed and optimized for reconfigurable modularity and the CNN is deployed in both software and hardware on an FPGA platform through a collaborative design. The results show that the proposed CNN accelerator has low power consumption, reduces resource consumption, and enhances system flexibility.

2. Background Knowledge

2.1 Convolutional Neural Network

Convolutional neural networks (CNNs) are feedforward artificial neural networks with a deep structure. Compared to traditional artificial neural networks, CNNs can directly process large amounts of pixel data from three-dimensional images and have good algorithm scalability and flexibility. The operation layers in a CNN typically consist of pooling layers, convolutional layers, fully connected layers, and activation layers. This study designs a CNN architecture consisting of two convolutional layers, two pooling layers, three activation layers, and two fully connected layers. The activation function layer uses rectified linear unit (RELU) activation function and the pooling layer uses maximum pooling operation. The network parameters are quantized to the 16-bit fixed-point design to reduce the number of parameters. The input to the network is handwriting digit images from the MNIST dataset. The network structure is shown in Figure 1, where C, P, R, and F represent convolutional layers, pooling layers, activation function layers, and fully connected layers, respectively.

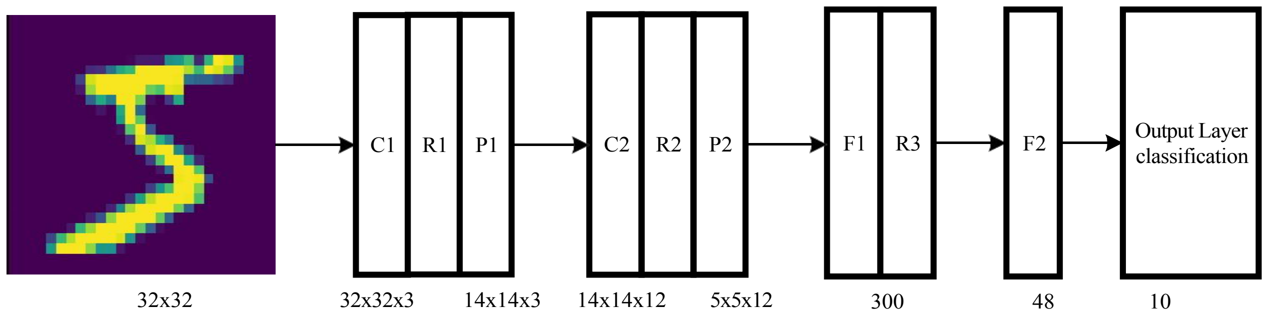

Fig 1. Structure diagram of CNN model

2.2 Dynamically Reconfigurable Technology of FPGA

Dynamically reconfigurable technology [22] in FPGA is a technique that enables hardware resources to be reconfigured during system operation. This allows the processor to dynamically allocate resources based on the changing requirements and demands of the application, improving processor performance and efficiency. In the design of dynamic reconfigurable CNN accelerators, this technology helps to quickly and efficiently allocate hardware resources of the processor when processing CNN models.

In practical applications, dynamically reconfigurable technology enables the system to dynamically adjust its hardware configuration in different application scenarios to meet different performance requirements. For example, when the processor needs to process a large amount of data, it can dynamically configure more hardware resources, and when processing a small amount of data, it can dynamically release hardware resources to reduce energy consumption.

Dynamically reconfigurable systems can be classified into two categories: complete reconfiguration and partial reconfiguration [19]. Complete dynamic reconfiguration changes the functionality of the system by reconfiguring all hardware resources through an FPGA. Partial dynamic reconfiguration (DPR) divides the whole FPGA area into static logic regions and dynamic logic regions. The static logic regions are the regions where the circuit structure is fixed, while the dynamic logic regions are the regions that can change by loading and unloading different hardware configuration files. The diagram for dynamic partial reconfiguration is shown in Figure 2, where the FPGA is divided into reconfigurable regions and static regions. The reconfigurable modules in the dynamic region are mapped to configuration files A1, A2, and A3 through register address mapping. During the operation of the system, different circuit configuration files can be dynamically configured from the configuration file library for different computation tasks [20]. By updating the hardware architecture dynamically in this way, the required hardware functions for the complete system can be constructed without interrupting the system, however, this technology is relatively complex both in design and implementation, and the stability of the configuration process is somewhat lacking.

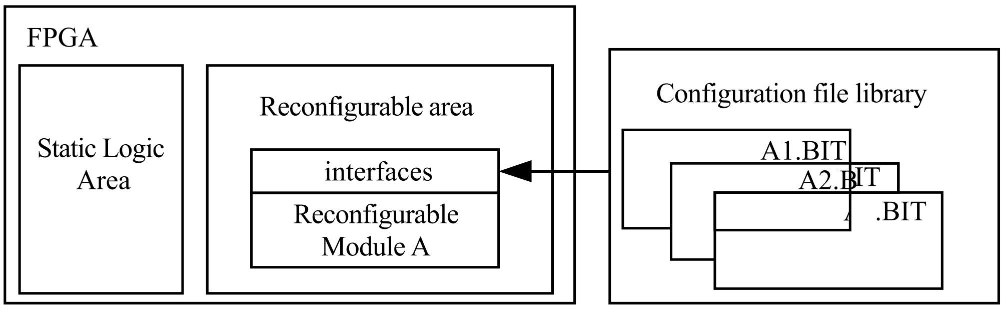

Fig 2. Schematic diagram of partial dynamic reconfiguration

3. Dynamic Reconfiguration Accelerator Design

3.1 Hardware Design

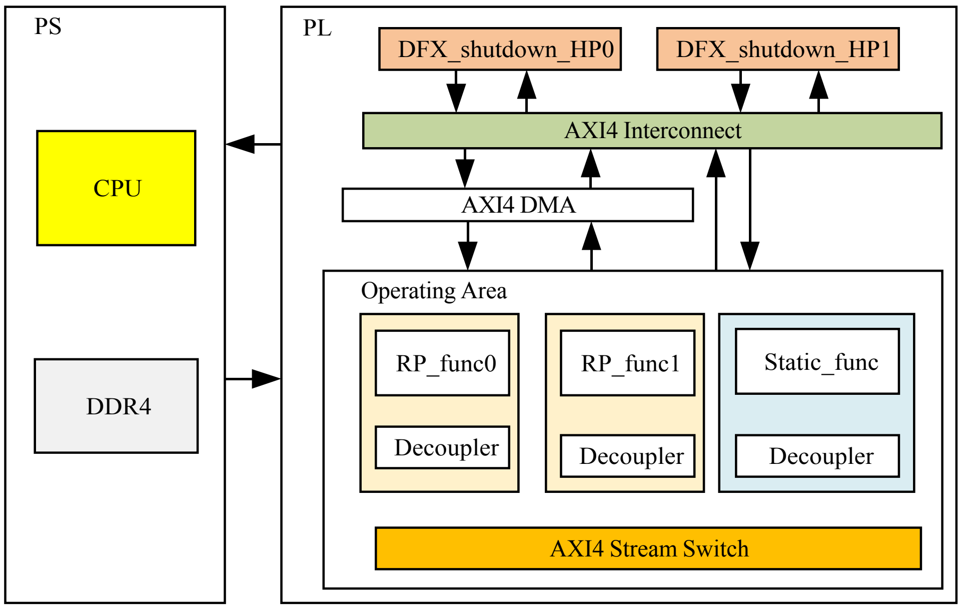

Fig 3. The overall architecture of the CNN accelerator

In this paper, a dynamically reconfigurable convolutional neural network accelerator architecture is proposed, and the overall architecture of this accelerator is shown in Fig. 3. The entire accelerator system is divided into a programmable logic (PL) side and a processing system (PS) side [21]. The hardware logic circuitry of the PL side is divided into dynamic and static logic regions, and its functional modules include the dynamic reconfigurable modular design of the convolutional activation module, the pooling module, the fully connected module, and the static modular design of the common basic operator operation acceleration module, and the direct memory access (DMA) module. The functional modules include the convolutional activation module, the pooling module, the fully connected module for dynamic reconfiguration modular design, and the acceleration module, the direct memory access (DMA) module, and the decoupler module for static modular design of the common basic arithmetic operations.

As depicted in Fig 3., the accelerator separates the functional modules under the Composable area into two dynamic reconfiguration partitioning functional module groups (RP_func0 and RP_func1) and a static logic module group (Static_func). The modules communicate through the AXI4-Lite bus and AXI4 Stream bus as the data communication interface, with the PS side using the AXI4 Lite bus to transmit control commands to the accelerator modules via the AXI4 Interconnect module. The AXI4-Stream bus is used to transmit control commands between RP_func0, RP_func1, the Static_func group, and the PS side through the DMA and AXI4 Stream Switch module at the PL side. The DMA controller facilitates high-speed data exchange between the accelerator and the external DDR4 for task data streams. Each computing module group is connected to the AXI4 Stream Switch module and communicates with the external DDR4 through the AXI4-Stream bus interface. To prevent logical confusion between the reconfigurable and static logic modules during operation, the design uses the Decoupler module for logical decoupling and the DFX_shutdown manager module to manage the AXI4-Lite and AXI4 bus interface for a safe accelerator operation.

The accelerator architecture in the design of this paper uses the central processor of the PS to interconnect and communicate with each module at the PL side and control the data flow through the memory interface. Due to a large number of operations between the operation layers, a high-speed interface design is used for each layer in order to improve the data transfer speed of each layer. In this paper, we are using the high-speed data channel of DMA to load the data in the PS memory into the buffer and transfer the data such as feature map, deviation, and weight to each operation accelerator on the FPGA in order through timing control, and when the previous operation step is finished, the result will be saved to the next input buffer.

In experiments, multiple calls to enable an unloaded DPR cluster block will generate a delay caused by loading bit streams, so it is important to avoid repeatedly enabling reconfigurable modules in the DPR cluster block. only one operator module can be enabled in a single use of the DPR region, and it will not be automatically unloaded after being enabled. Therefore, in this paper, we put the convolutional, pooling, and fully connected arithmetic modules with the same interface but different sizes into different DPR cluster blocks and enable the modules, design the modules other than arithmetic modules as static logic, and put the pooling arithmetic module and fully-connected arithmetic module as reconfigurable modules into the reconfigurable partitions of RP_fun0 and RP_fun1 respectively. As the above design, it can make the system used in a non-specific network layer block or network architecture, improve the system flexibility while reducing the delay caused by loading the DPR computing module, improve the system computing performance, and reduce the power loss caused by occupying too much static logic area.

3.2 Hardware Design

The reconfigurable modules of the accelerator in this paper mainly contain the convolutional operation accelerator, pooling operation accelerator, and fully connected operation accelerator. In order to improve the integration of the reconfigurable modules, the convolutional operation layer and the activation function layer set become the convolutional activation layer, and similarly the fully connected layer and the activation function layer set become the fully connected activation layer. As shown in Figure 2, this paper divides the FPGA into a static logic region and two dynamic reconfiguration regions, and each dynamic reconfiguration region is spatially multiplexed with three reconfigurable modules, thus forming a reconfigurable module group. When reconfigurable modules are assigned to different reconfiguration regions, the same reconfiguration region principle needs to be followed, i.e., the number of on-chip hardware resources occupied by reconfigurable modules should be as close as possible to reduce the waste of hardware resources [22]. The top-level design block diagram of the reconfigurable module group is shown in Fig. 4. The accelerator system selects the reconfigurable module into the execution process through the AXI4 Stream Switch module, and in order to prevent the logic mixing between the dynamic logic region and the static logic region from making serious errors in the system, the reconfigurable module is decoupled and designed using the Decoupler module to decouple the logic of the reconfigurable module. Decoupler module is used to decouple the logic of the reconfigurable module.

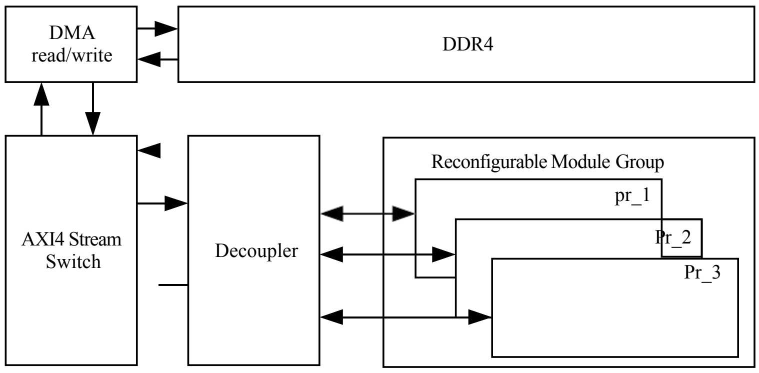

Fig 4. Top-level design block diagram of the reconfigurable module group

3.3 Operation Unit Optimization Strategy

Current optimization methods commonly used to improve the performance of accelerator image processing in embedded edge hardware include loop pipelining and loop unfolding [7]. The loop unfolding method is to unfold the loop iterations and let the parallel operations within the loop, which can be very good to improve the operation speed, but it will significantly increase the hardware resource consumption, which can be used to increase the resource utilization of PL in large-scale operations. Loop pipelining, on the other hand, increases system throughput by repeatedly executing operations from different loop iterations with only a small increase in hardware resource consumption.

The hardware interface design of each CNN model adopts AXI4 Stream bus interface design, which transfers data from memory to CNN IP core through DMA, and then transfers the processed data in back to DDR4 again, with a maximum transfer rate of 2133MB/s, which can provide ample bandwidth and effectively enhance the system. The algorithm topology uses a combination of cyclic pipelining and cyclic unfolding optimization design, as shown in Algorithm 1, with a fully parallel design for the parallel operation part and a fully pipelined design for each operation layer, in order to shorten the delay line (TDL), reduce the operation delay, and improve the accelerator throughput. Since the convolutional layer in the network consumes a lot of computational resources and has a long delay line length, a pipeline with a loop starts interval value of one, a specified number of iterations for loop execution, and a specified use of computational resources are used to shorten the delay line length as much as possible while making full use of computational resources to improve the system performance. The optimization methods of the pooling operation accelerator and the fully connected accelerator are the same as those of the convolutional operation accelerator.

Algorithm 1. Parallel operation optimization pseudo-code

```
Input: InputFeature[too][trr][tcc],
       Weighs[Tcout][Tcin][H][W];
Output: OutFeature[Tcout][Tcout][Tyout];
```

```
FOR(i=0;i<H;i++){
 FOR(j=0;j<W;j++){
   FOR(trr=row;trr<min(row+Tr,R);trr++){
     FOR(tcc=col;tcc<min(col+Tm,C);tcc++){
#PRAGMA HLS PIPLINE II=1 // Pipeline
       FOR(too=to;too<min(to+Tn,M);tii++){
#PRAGMA HLS UNROLL // Parallelization
         FOR(tii=ti;tii<min(ti+Tn,N);tii++){
#PRAGMA HLS UNROLL // Parallelization
LOOP:    OutFeature[too][trr][tcc] +=\
         Weights[too][tii][i][j]*\
         InputFeature [tii][S*trr+i][S*tcc+j];
}}}}}}
```

3.4 Design Flow

The design flow of the CNN accelerator is outlined in Figure 5. The process starts with building the CNN algorithm model on the Tensorflow deep learning framework [23]. The model's parameters are then trained on the MNIST dataset and quantized to 16-bits fixed-point, saved as binary files. PYNQ-Kria, a software-hardware collaboration framework, is used as the embedded operating system and hardware deployment environment on the XCK26-SFVC784 heterogeneous ARM and FPGA platform. The IP soft cores for the convolutional activation module, pooling module, and fully connected module are designed and packaged using the Vitis HLS development tool [24]. Hardware design optimization and module reusability are performed using the Vivado tool. The overall FPGA hardware design of the CNN accelerator on the PL (Programmable Logic) side is done using Vivado. Finally, the PS (Processing System) side, which acts as the host side, uses the Jupyter Notebook development platform to call the IP softcore on the PL side and configure the network parameters, completing the software application of the designed CNN accelerator.

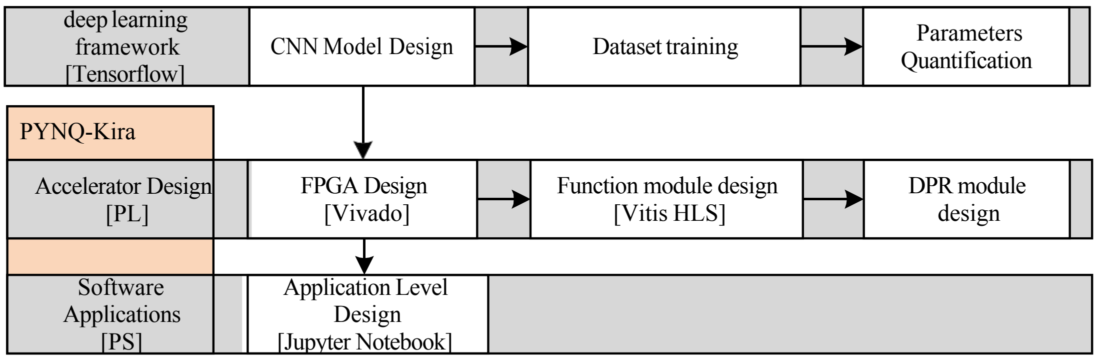
deep learning framework [Tensorflow]
CNN Model Design
Dataset training
Parameters Quantification
PYNQ-Kira
Accelerator Design [PL]
FPGA Design [Vivado]
Function module design [Vitis HLS]
DPR module design
Software Applications [PS]
Application Level Design [Jupyter Notebook]

Fig 5. Design process of CNN accelerator

3.5 System Scheduling Flow

The scheduling flow chart of the CNN accelerator system in this paper is shown in Figure 6. It uses the PetaLinux [25] operating platform to deploy the soft and hard collaborative framework PYNQ-Kira as the embedded Linux operating system. the FPGA is controlled by the configuration parameter commands input from the upper computer side, that is, the PS side, to complete the operation start-up and switching layer by layer according to the convolutional neural network layer.

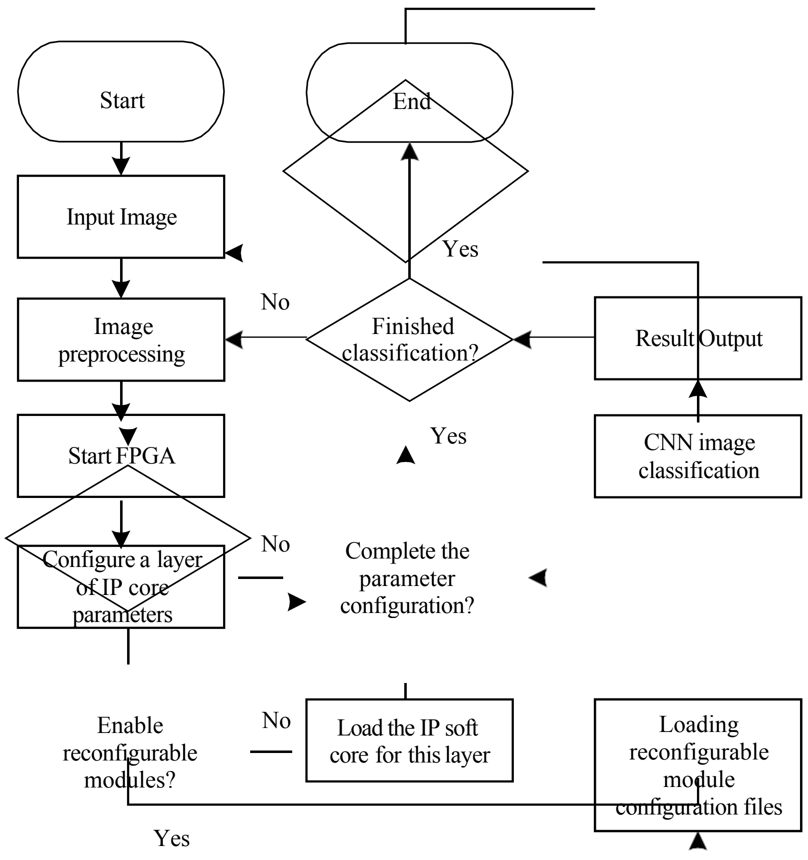
Start
Input Image
Image preprocessing
Start FPGA
Configure a layer of IP core parameters
Enable reconfigurable modules?
Yes
No
Load the IP soft core for this layer
Loading reconfigurable module configuration files
No
Complete the parameter configuration?
Yes
CNN image classification
Result Output
Finished classification?
No
Yes
End

Fig 6. Procedure of CNN accelerator system

4. Results

In order to verify the CNN accelerator in this paper, the development tools Vivado (2020.2) and Vivado HLS (2020.2) of AMD-XILINX are used to complete the engineering design and hardware module design and synthesis verification on the PL side, and then the PYNQ-Kira framework is deployed through PetaLinux operating platform for soft and hard co-design, and the already generated The chip model of the KV260 hardware platform is XCK26-SFVC784, the CPU of its PS side is ARM Cortex A53 CPU, and the off-chip high-speed memory is DDR4, the image to be tested is imported into a 16G SD card, and then the hardware platform reads the image and enables the FPGA to complete the convolutional neural network The image classification inference operation of the convolutional neural network is completed by the FPGA.

The CNN accelerator proposed in this paper, the inference network adopts the CNN architecture of Fig. 1, and the engineering design and construction of the accelerator are finally completed after the design flow shown in Fig. 3. The actual hardware resource consumption of the PL side, after synthesis and verification by Vivado, is shown in Table 1. If the CNN accelerator in this paper is designed with a completely static circuit, its actual hardware resource consumption is shown in Table 2.

Table 1. Hardware resource consumption of CNN accelerator

Resource Type	FF	LUT	BRAM	DSP
Usage	17967	21412	59	54
Resource Quantity	234240	117120	144	1248
Utilization (%)	7.67	18.28	40.97	4.09

Table 2. Hardware resources consumption of Completely static design of the CNN accelerator

Resource Type	FF	LUT	BRAM	DSP
Usage	21848	31874	115	102
Resource Quantity	234240	117120	144	1248
Utilization (%)	9.32	27.21	79.86	8.17

The data comparison shows that with the same hardware platform, the resource consumption of the accelerator in this paper is significantly lower than that of the accelerator with a completely static design, using 17.76%, 32.82%, 48.70%, and 47.01% less FF, LUT, BRAM, and DSP, respectively. Since the accelerator in this paper utilizes dynamic reconfiguration and reuses other reconfigurable modules in the dynamic reconfiguration region space in advance so that it can dynamically load reconfigurable modules, the area used for system logic resources and resource consumption are reduced substantially.

Table 3. Compare the results of different CNN accelerators

Type	Device	Data Precision	Clock Frequency /MHz	Power/W	Throughput /GOPS	Processing Time on PL/ms	Energy Efficiency (GOPS/W)
Lit.[9]	XC7Z020	Fixed16	100	5.04	**20.53**	--	4.07
Lit.[10]	ZU3EG	Fixed16	170	3.55	31	4.6	8.73
Lit. [11]	ZU3EG	Fixed16	169	4.7	51	0.174	**10.85**
Lit. [22]	XC7Z020	Fixed16	100	**7.8**	18.35	20.3	2.35
This Work	XCK26	Fixed16	200	2.92	26.2	**0.142**	8.97

Table 3 compares the power consumption and performance of the accelerator in this paper with some other accelerators. From Table 3, it can be seen that the CNN accelerator designed in this paper shows obvious advantages in low power consumption and performance compared with those in the literature [9, 10, 11, 22], and its power consumption is reduced by 42.06%, 17.75%, 37.87% and 62.56%, respectively; its comparison with the accelerators in the literature [10, 11, 22], the accelerator in this paper spends less time on network inference in PL processing time reduction ratios of 3139.37%, 22.54% and 14195.77%, respectively; its energy efficiency ratios compared with those of the literature [9,10,22] are improved by 120.39%, 2.75% and 281.70%, respectively.

5. Conclusion

In order to improve the flexibility of deploying convolutional neural networks on embedded edge computing devices and reduce their hardware resource consumption and operational power consumption, this paper designs a dynamic reconfigurable accelerator for convolutional neural networks, which significantly reduces operational power consumption and resource consumption through hardware and software co-design and dynamic reconfiguration design and optimization. The accelerator supports reconfigurable module extensions and can be applied to different network architectures. The experimental results show that the power consumption of the accelerator is 2.92W, which is 42.06%, 17.75%, 37.87%, and 62.56% lower than the accelerator of the current study, and the hardware resources FF, LUT, BRAM, and DSP are 17.76%, 32.82%, 48.70%, and 47.01% lower than the accelerator of the completely static circuit, respectively. 47.01%. The next step will be to explore the optimization strategy of the dynamically reconfigurable accelerator and extend the deployed CNN algorithm to a CNN-based lightweight target detection algorithm to further improve the overall performance of the dynamically reconfigurable hardware accelerator for more complex applications.

References

[1] REUTHER A, MICHALES P, Jones M, et al. AI and ML Accelerator Survey and Trends [C] //2022 IEEE High Performance Extreme Computing Conference, Waltham, USA: IEEE,2022:1-10.

[2] ZHANG Jiehe, ZHANG Feng, Xie Min, et al. Design and Implementation of CNN Traffic Lights Classification Based on FPGA[C] //2021 IEEE 4th International Conference on Electronic Information and Communication Technology, Xi'an China, IEEE, 2021: 445-449.

[3] PESTANA D, MIRAND P R, LOPES J D, et al. A Full Featured Configurable Accelerator for Object Detection With YOLO[J]. IEEE Access, 2021, 9: 75864-75877.

[4] Chen Y H, Fan C P, Chang R C. Prototype of Low Complexity CNN Hardware Accelerator with FPGA-based PYNQ Platform for Dual-Mode Biometrics Recognition[C] //2020 International SoC Design Conference, Yeosu, Korea, IEEE, 2020:189-190.

[5] SRIVASTAVA H, SARAWADEKAR K, DEPTHWISE A. Separable Convolution Architecture for CNN Accelerator[C] //2020 IEEE Applied Signal Processing Conference, Kolkata, India: IEEE,2020:1-5.

[6] SHI Yali, Gan Tong, JIANG Shaobo, et al. Design of Parallel Acceleration Method of Convolutional Neural Network Based on FPGA[C] //2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics, Chengdu, China: IEEE,2020:133-13.

[7] ZHANG Chen, LI Peng, SUN Guanyu, et al. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks[C] //Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey California USA: ACM,2015:161–170.

[8] ZHANG Yao, DUAN Qing, SHA Guanglin, et al. Research on Dynamic Reconfiguration of Distribution Network Based on Reconfiguration Effectiveness[C] //2021 4th International Electrical and Energy Conference, Wuhan, China: IEEE,2021:1-6.

[9] ZHOU Zhengjie, LIU Yumei, XU Yidong. Design and implementation of YOLOv3-Tiny accelerator based on PYNQ-Z2 heterogeneous platform[C] //2020 4th International Conference on Electronic Information Technology and Computer Engineering, New York, NY, USA: ACM,2020:1097–1102.

[10] KIM B, HORSTED S, EJNAR L. A scalable and efficient convolutional neural network accelerator using HLS for a system-on-chip design[J]. Microprocessors and Microsystems,2021,87(2):1-18.

[11] ALHUSSAIN A, LIN M. Hardware-Efficient Template-Based Deep CNNs Accelerator Design[C] //2022 IEEE International Conference on Networking, Architecture and Storage, Philadelphia, PA, USA: IEEE, 2022: 1-4.

[12] SHANG Qianyi, CHEN Lijun, TONG Ruoxiong. Hardware/Software Co-design for Evolvable Hardware by Genetic Algorithm[C]. //2020 IEEE International Conference on Artificial Intelligence and Information Systems, Dalian, China, IEEE, 2020: 306-309.

[13] ALOYSIUS N, GEETHA M.A review on deep convolutional neural networks[C] //2017 International Conference on Communication and Signal Processing, Chennai, India: IEEE,2017:0588-0592.

[14] ANG Xiao, DENG Junyong, XIE Xiaoyan. Design and implementation of reconfigurable CNN accelerator [J]. Transducer and Microsystem Technologies, 2022,41(02):82-85,89. (in Chinese)

[15] I Jing, ZHANG Xuxin, JIN Jie. Fast license plate recognition system based on dynamic reconfiguration of FPGA[J]. Transducer and Microsystem Technologies,2019,38(12):69-72. (In Chinese)

[16] BACHTIAR Y, ADIONO T. Convolutional Neural Network and Maxpooling Architecture on Zynq SoC FPGA[C] //2019 International Symposium on Electronics and Smart Devices, Badung, Indonesia: IEEE, 2019: 1-5.

[17] YOYSSEF E, ELSIMARY H A, EL-MOYSY M, et al. Energy-Efficient Precision-Scaled CNN Implementation with Dynamic Partial Reconfiguration[J]. IEEE Access,2022, 10,95571-95584.

[18] YU Zijian, MA De, YAN Xiaolang, et al. FPGA-based Accelerator for Convolutional Neural Network[J]. Computer Engineering, 2017, 43(1): 109-114, 119.

[19] SUNKAVILLI S, CHENNAGOUNI N, YU Q. DPReDO: Dynamic Partial Reconfiguration enabled Design Obfuscation for FPGA Security[C] //2022 IEEE 35th International System-on-Chip Conference, Belfast, United: IEEE,2022:1-6.

[20] UAN Fuli. Convolutional neural network accelerator based on dynamic hardware recongfiguration[D].

Hefei, CHINA: University of Science and Technology of China,2021.

[21] WANG Lin, AO Tianyong, FU Liu, et al. Design of a YOLO Model Accelerator Based on PYNQ Architecture[C] //2022 International Conference on Machine Learning and Intelligent Systems Engineering, Guangzhou, China: IEEE,2022:15-18.

[22] BAO D, HAO L. Research on Dynamic Reconfigurable Convolutional Neural Network Accelerator[J]. Journal of Physics,2021,1952(3):1-6.

[23] SATYAVOLU S, BAGUBALI A. Implementation of TensorFlow and Caffe Frameworks: in View of Application[C] //2019 International Conference on Vision Towards Emerging Trends in Communication and Networking, Vellore, India, IEEE,2019:1-4.

[24] KHVATOV V M, ZHELEZNIKOV D A. Development of an IP-cores Libraries as Part of the Design Flow of Integrated Circuits on FPGA[C]. //2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, Moscow, Russia, IEEE, 2021:2686-2691.

[25] ALMEIDA T B, PEDRINO E C, FERNANDES M M. Complex Morphological Filtering for Serial, Parallel, GPU, SoC, PetaLinux and FPGA Execution[J]. IEEE Latin America Transactions,2020, vol.18 (10):1675-1682.