Design and Implementation of RISC-V Based Convolutionally Accelerated Processors

Zhekai Cui

School of Information Science and Technology, Shandong University, Qingdao, 266237, China
cui.zhekai@mail.sdu.edu.cn

Abstract. Convolutional neural networks (CNNs) have become widely used in a variety of application scenarios due to the rapid development of deep learning and artificial intelligence technologies. However, traditional general-purpose processors face performance bottlenecks when dealing with large-scale convolutional operations. Specifically, the Reduced Instruction Set Computer V (RISC-V) open-standard design is utilized in many publicly accessible implementations and has been warmly embraced by the international community of researchers and business users. In this paper, the author propose a system-on-chip (SOC) design depending on the RISC-V instruction set architecture, which significantly improves the efficiency of convolutional operations by extending the custom instruction set and integrating a dedicated AI computation unit (AI Core). The architectural design, module partitioning, sub-module implementation, and simulation results of the processor are described in detail in this paper. According to the experimental results, the design maintains high accuracy when running the Yolv4 small model while achieving a 10x speedup in convolutional operations when compared to conventional CPU serial operations.

Keywords: RISC-V, Accelerator, System-on-Chip.

1 Introduction

Convolutional Neural Networks (CNNs) have made notable strides in image identification, target detection, and related fields thanks to the quick growth of AI technology. However, convolutional operations within CNNs require intensive computational resources. This poses significant challenges for conventional general-purpose processors, especially in edge computing environments. This is particularly the case in scenarios where there is a high real-time requirement. To address this challenge, dedicated hardware accelerators (e.g., FPGAs, ASICs) have gradually become a research hotspot, and RISC-V, an open source instruction set architecture (ISA), is gradually emerging in the field of AI hardware acceleration due to its high flexibility and scalability. In contrast to conventional x86 and ARM architectures, RISC-V enables developers to customise both the instruction set and the hardware design according to the particular requirements of each application, thus allowing the creation of highly efficient convolutional acceleration units.

J. Iqbal (ed.), *Proceedings of the 2025 2nd International Conference on Mechanics, Electronics Engineering and Automation (ICMEEA 2025)*, Advances in Engineering Research 272,
https://doi.org/10.2991/978-94-6463-821-9_43

The openness of RISC-V is a significant strength, as it allows developers to customise and extend the instruction set freely, in contrast to traditional closed instruction set architectures. This flexibility endows RISC-V with considerable potential in the domain of AI hardware acceleration. Moreover, RISC-V supports custom instruction extensions, enabling developers to incorporate specialised instructions that align with specific application requirements. For instance, specialized matrix multiplication instructions with SIMD support and optimized data handling instructions can be designed specifically for convolutional operations, potentially improving computational efficiency by up to 5×. The simplicity of the RISC-V architecture confers upon it a substantial advantage in power consumption control. This feature is particularly salient for edge computing devices, where low power consumption is a key metric, and it is therefore ideal for edge AI acceleration.

In the context of neural networks, convolutional operations are recognised as being among the most time-consuming components, particularly in the processing of high-resolution images and multi-channel data. The inherent limitations of traditional general-purpose processors in terms of parallel computing power have been a significant hindrance in the efficient execution of such operations. For example, Tianshi Chen et al. designed an accelerator for large-scale CNNs and DNNs. Their design achieved a 117.87x speedup compared to a 128-bit 2GHz SIMD processor. The accelerator executed 452 GOP/s while maintaining a small footprint of 3.02 mm2 and consuming only 485 mW, resulting in a 21.08x reduction in total energy consumption [1]. In February 2015, Chen Zhang employed the Roofline model to analyze various optimization techniques, revealing that memory bandwidth often becomes the primary bottleneck in CNN acceleration [2].

The proliferation of the Internet of Things (IoT) and the advent of edge computing have led to an increasing demand for AI applications to be executed on edge devices. These devices, characterised by their limited resources and stringent requirements for computational efficiency and power consumption, present a considerable challenge to the execution of such applications. RISC-V stands out as a notable solution, offering a combination of flexibility and low power consumption, thereby enabling the custom design of instruction sets based on specific application requirements. By implementing dedicated convolutional acceleration units, RISC-V can achieve parallel convolutional computing capabilities that significantly surpass those of conventional processors while maintaining low power consumption.

This study proposes a novel hardware acceleration solution that combines RISC-V's customizable instruction set with dedicated convolutional units, specifically designed to optimize AI computations while maintaining low power consumption. The study designs a 3-stage pipelined, single-launch CPU by customising the instruction set, connecting via the AXI4 bus, and incorporating convolutional preprocessing and matrix multiplication units into the CPU execution link to enhance the efficiency of convolutional computation. The convolutional operation is implemented using pulsed arrays, and the matrix multiplication units are implemented by cascading PE units to improve computational efficiency with less resource utilisation. In addition, a new solution is provided for its added peripheral controllers, including GPIO, SPI, UART, and TIMER. Furthermore, the data path is optimised, the data handling overhead is reduced, and the overall performance is improved. Consequently, edge computing devices are provided with highly efficient and low-power AI acceleration solutions.

2 Literature Review

2.1 RISC-V ecosystem

By increasing need for local data processing, the Internet of Things (IoT) huge data production at the network edge has significantly expedited the development of edge computing and edge AI systems. Developers can now more effectively construct information-physical systems near the data source because to the widespread use of embedded devices, sensors, and actuators. The edge computing paradigm emphasises the use of energy-efficient embedded systems where resource-constrained hardware in complex application scenarios requires more processing power to run signal processing and machine learning algorithms.

In order to satisfy the need for effective accelerators in terms of power consumption and computational performance (measured in GOPS/W), specialized hardware processors have been designed. Traditionally, embedded CPU cores focus on general computing tasks, while GPUs excel at big data analysis and CNN training. Meanwhile, FPGAs and ASICs have emerged as solutions that offer improved power consumption and computational efficiency.

However, designers are now investigating large-scale open-source processor designs as a result of research and innovation into more sophisticated, domain-specific hardware accelerators, and the RISC-V Instruction Set Architecture (ISA) standard has been created. With its open-source, flexible and scalable features, the RISC-V architecture has attracted widespread attention in areas such as the Internet of Things, where it is expected to meet the demand for low-cost, energy-efficient devices and drive the development of related technologies [3].

2.2 FPGA accelerator

Convolutional neural networks have seen widespread implementation across a range of domains. However, conventional convolution operations are computationally intensive and traditionally rely on high-performance computing platforms such as GPUs, FPGAs, or specialized accelerators. The high power consumption of GPUs poses a significant constraint on their deployment in embedded devices. For accelerator design, Lin Bai reduces the computational load and parameter requirements of models such as MobileNetV1 and MobileNetV2 through deeply separable convolution. CNN operations are processed through MME arrays and data is managed to optimise bandwidth and latency. Experiments on the Arria 10 SoC development kit with a 16-bit quantisation strategy show that the accelerator running at 133MHz achieves 170.6 GOPS and processes 266.6 frames per second, which is 20 times faster than the CPU implementation [4].

Based on the findings from these two papers, this paper can see both the practical applications of RISC-V technology and the feasibility of implementing parallel CNN arrays on FPGAs. The second article focuses on FPGA accelerators and provides a strong demonstration of the feasibility of executing models on FPGAs by experimenting with quantised models on the Arria 10 SoC development kit. This lays the foundation for further research on the integration of RISC-V and FPGAs in machine

learning, and inspires us to further think about how to make full use of the advantages of both, optimise the existing design, expand the application scenarios, and improve the overall effectiveness and performance.

3 Implementation of RISC-V Accelerator

3.1 SOC architecture design

The design incorporates a RISC-V CPU core along with a number of peripheral modules and interfaces to create a System-on-Chip (SOC) based on the RISC-V instruction set architecture. The SOC system communicates between modules via the AXI bus protocol and supports functions such as memory operations, interrupt management, timers, GPIOs, SPIs and UARTs. Peripheral and interface modules including AXI Cross Bar 1x2, AXI Protocol Converter, Extern Interrupt, Config Reg, TIMER, DDR3, GPIO, SPI, UART Controller are used to support system communication, memory, timing and external devices. Memory, timing and external device interaction functions. The overall structure is shown in Fig. 1.

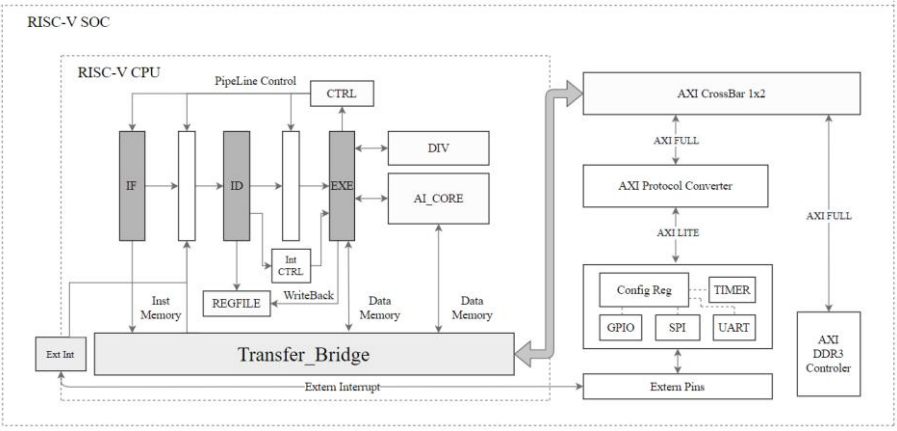

Fig. 1. SOC General Architecture Diagram (Photo/Picture credit: Original).

3.1.1 RISC-V CPU core.

Includes Pipeline Control, Instruction Fetch (IF), Instruction Decode (ID), Execution (EXE), ALU Core (AL CORE), Interrupt Control (Int CTRL), and Instruction Memory, Register File (REGFILE), Write Back and Data Memory submodules. These modules work together to perform basic computation and control functions.

3.1.2 AI Core module.

As a key component of the SOC, the AI Core accelerates convolution and pooling operations through custom instructions and adopts a pulsed array structure consisting

of multiple Processing Element (PE) units to support highly efficient matrix multiplication calculations.

3.2 Pipeline design

The design implements a three-stage pipeline: fetch (instruction retrieval), decode (instruction parsing and register operations), and execute (computation and memory operations), and pipeline operation is paused by the CTRL module to satisfy the operation of large-cycle instructions. The SOC supports the RV32IM instruction set and is able to implement C programming using the RISC-V compilation chain, it is designed with the AXI full bus to support high concurrent reads with high bandwidth, and it extends the original instruction set with custom instructions to adapt its convolution and pooling acceleration operations.

At the instruction fetch stage, the CPU retrieves 32-bit RISC-V machine code. The decoding stage then determines the operation type and parameters based on this code, and finally, through the execution stage, according to the specific operation of the instruction obtained from the decoding, to complete the access to memory and write back to the register group of the relevant operations, for the division and convolution, pooling operations will call the DIV and AI Core modules for division, convolution and pooling operations.

For certain instructions, such as the operation of jump instructions and long cycle instructions, pipeline suspension is used here to resolve the occurrence of adventures (Fig. 2).

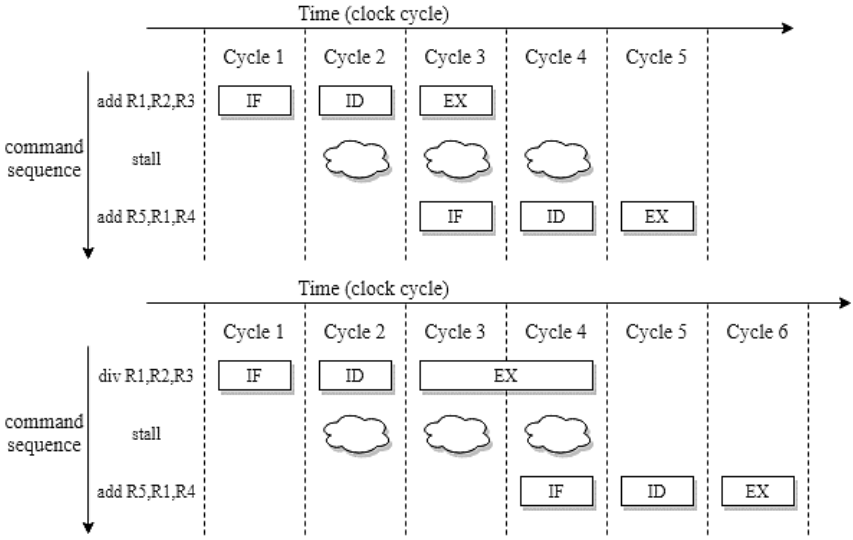

Fig. 2. CPU Pipeline Structure (Photo/Picture credit: Original).

3.3 Accelerator architecture design

The AI Core computing unit has been engineered to accelerate convolutional operations through matrix acceleration computation. The implementation of this technology is comprised of the following steps: (1) Matrix conversion: the convolution operation is converted to matrix multiplication by img2col operation, and the convolution kernel and input feature map are unfolded into a matrix. (2) Pulsed array computation: The AI Core contains multiple PE units arranged in a systolic array structure, where each PE performs partial matrix multiplication operations in parallel, enabling efficient 64x64 matrix computations. Through parallel computation, AI Core is able to complete 64x64 matrix multiplication computation in a shorter cycle. (3) Chunked computation: In order to process larger scale feature maps, the concept of chunked computation is employed. Initially, the output channel is processed in chunks, then the input feature map is chunked, with each operation processing multiple output channels. The results are then spliced and accumulated through the accumulation module.

The AI Core computing unit implements matrix accelerated computation, parses the corresponding instruction type through EX, and completes the chunking of the matrix to regulate the whole system through Control Unit. The complex convolution operation is converted into matrix operation through img2col, then the matrix multiplication is calculated through the pulsed matrix, and the matrix chunking multiplication is achieved through the accumulation module, after which pooling is performed to transfer the data back to memory. DDR inputs to BRAMA and BRAMB are initially cached by the accelerator. To finish the calculation, it then uses TDM to write the results after performing ping-pong operations (Fig. 3).

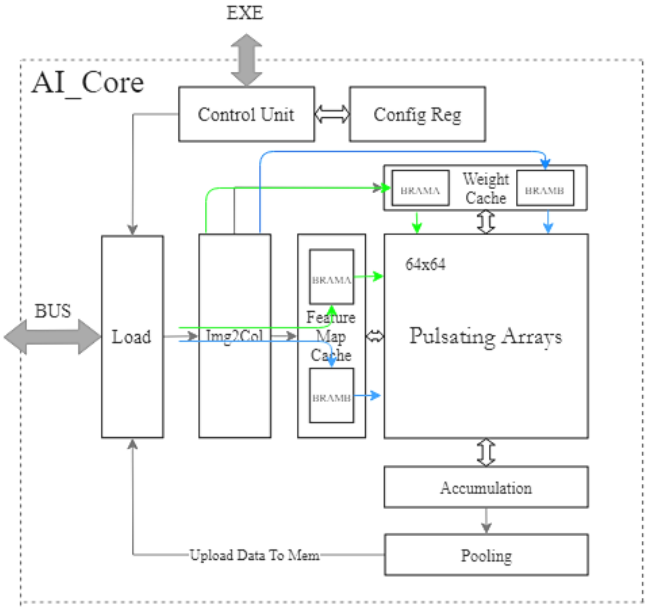

Fig. 3. Ai Core block (Photo/Picture credit: Original).

3.4 Matrix multiplication

3.4.1 I/O problem.

I/O Problem (Input/Output Problem) is a performance bottleneck or inefficiency in a computer system caused by an input/output operation (e.g., reading, writing, or transferring data). I/O problems usually occur between computing resources (e.g., CPU, GPU) and storage devices (e.g., hard drives, SSDs) or network devices.

It is hypothesized that, regardless of the special-purpose system's operating speed, the maximum achievable rate is limited to 5 million operations per second in a system with a high I/O bandwidth of 10 million bytes per second between a host and a special-purpose system. This is predicated on the assumption that a minimum of two bytes are read from or written to the host for each operation (Fig.4.). However, it is important to note that orders of magnitude improvements in throughput are only possible if multiple computations are performed per I/O access. The repetitive use of a data item necessitates its storage within the system for a sufficient length of time. Therefore, the I/O problem is not only related to the available I/O bandwidth, but also to the available memory internal to the system. The challenge, therefore, lies in the effective organisation of computations and the allocation of memory structures, ensuring that time required for computation is balanced with that of I/O [5].

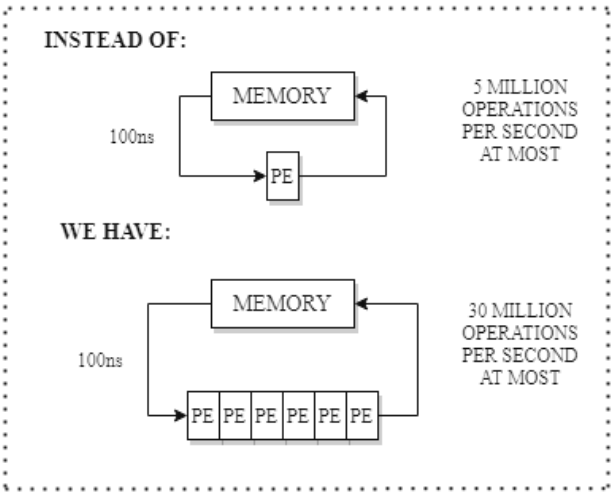

Fig. 4. Basic principle of a systolic system [5]

3.4.2 Systolic architecture.

I/O issues are crucial in computation systems. These issues become particularly challenging when dealing with systems that have limited specialized capabilities. In such cases, it becomes necessary to decompose the computation in question. The sequential execution of Sub-Computations can necessitate significant I/O operations for the storage or retrieval of intermediate results [5].

Cells within a systolic system are usually interconnected to form arrays or trees because a systolic system is made up of a collection of interconnected cells, each of which is capable of carrying out a simple operation. This property offers significant advantages in terms of design and implementation when compared with complex communication and control structures. A systolic system's information flow is pipelined, and only at the "boundary cells" does communication with the external environment take place. Only cells on the array boundaries, for example, are allowed to serve as the system's input/output ports in a systolic array.

A 64×64 scale pulsating array was designed (Fig. 5). Each node is characterised by a PE unit composition, and the structure, accumulation, and output are facilitated by the accumulation unit. It has been demonstrated that matrix multiplication can be achieved in 128 cycles by employing a 64x64 sized pulsed array, thereby ensuring high efficiency with minimal resource utilisation.

3.5 Pooling IP core design

The fundamental objective of the pooling layer is to lower the amount of computation and the number of parameters while effectively conserving critical properties, thus enhancing computational efficiency. The IP cores can be configured in steps through the AXI4 bus write registers to enable massively parallel processing of pooling operations, and the module adapts 2x2 sized average pooling and maximum pooling [6].

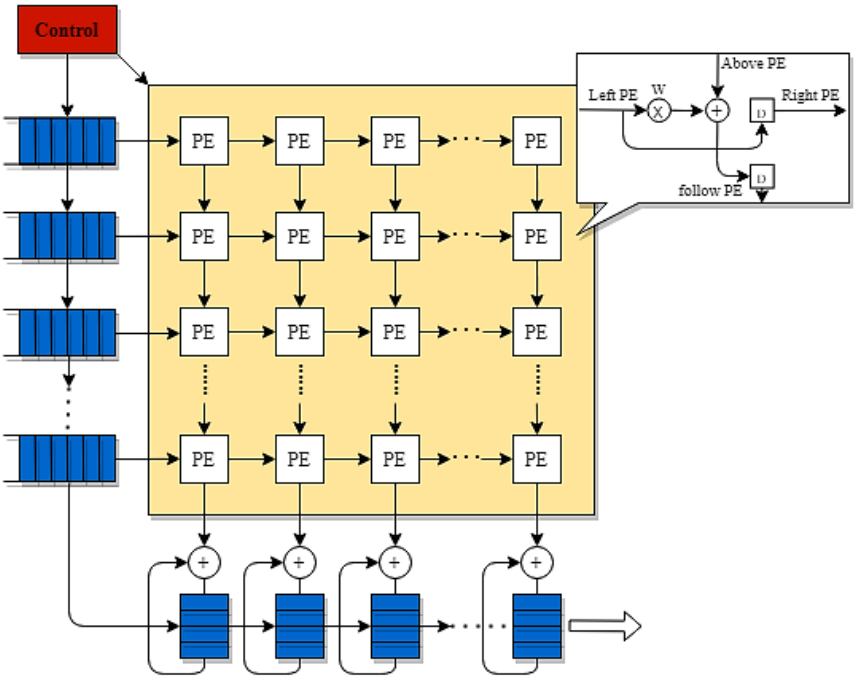

Fig. 5. Pulsed Array Multiplier Unit [7].

4 YOLOv4 Tiny Model Deployment

4.1 Advantages of the yolov4 tiny model

YOLOv4 Tiny is a lightweight version of YOLOv4 that has been designed for real-time target detection tasks on devices with limited computing resources. A comparison of the two models reveals that YOLOv4 Tiny exhibits a significantly higher processing speed than YOLOv4. Furthermore, the number of parameters required for YOLOv4 Tiny is approximately one-tenth that of YOLOv4. The model's small size and fast speed can be leveraged to a certain extent to accelerate the model's running process prior to quantization [8].

4.2 Network structure of the yolov4 tiny model

The CSPDarknet53 network in Yolov4 is replaced with the CSPDarknet53-tiny network as the main network in the Yolov4-tiny technique. Instead of using the ResBlock module in the residual network, the CSPDarknet53-tiny network uses the CSPBlock module from the cross-stage partial network. The CSPBlock divides the feature map, combining the parts via cross-stage residual edges, thereby allowing gradient flow in two paths to enhance gradient information correlation difference. This enhances the convolutional network's learning ability compared to the ResBlock, though it raises computation by 10%-20% and improves accuracy. To reduce calculation, it removes high-computation bottlenecks in the CSPBlock. Thus, Yolov4-tiny improves accuracy with constant or reduced computation [9] (Fig. 6).

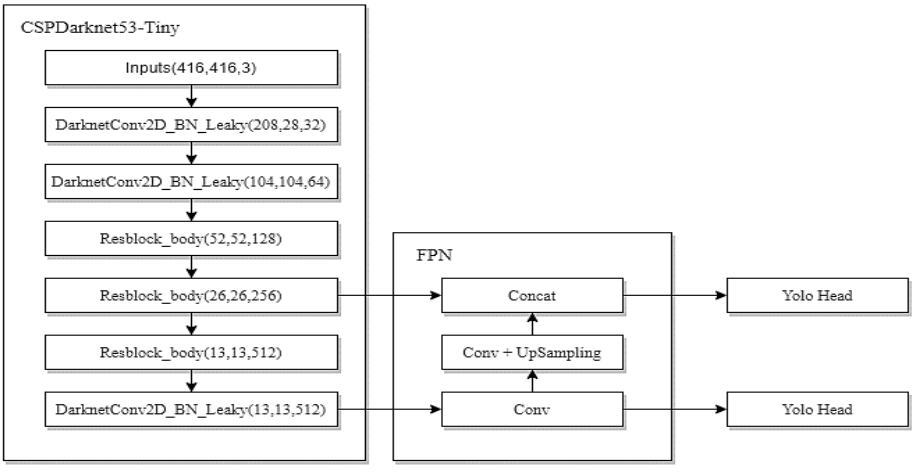

Fig. 6. Network structure [9]

5 Results and discussion

This section of the article presents a summary of the results obtained from the experimental investigation into the SoC acceleration structure in the context of the specific application. In this research, object detection studies were performed using the COCO dataset and the Yolov4-Tiny model. The SoC accelerator was deployed on an FPGA (XC7A200TFBG676-1 main chip) for performance evaluation and testing purposes.

The average processing time was calculated using 100 images from the COCO dataset, with results shown in Table 1, comparing the results calculated on CPU Cortex-A9, It is clear from the experimental data that while the optimized model's running speed is almost thirty times faster than the unoptimized model's, the optimized model's inference accuracy is essentially unaltered. In other words, the optimised system will significantly increase the model inference speed, while the loss of accuracy is almost negligible, which provides a great advantage and convenience for the application of neural network models. RISC-V SoCs enhance the efficiency of convolutional computation with optimal energy consumption. This makes them ideal for edge computing applications.

Table 1. CPU Cortex-A9 and FPGA-Based Accelerator speed and average inference accuracy.

Execution System	Average Inference Speed	Average Inference Accuracy	Ref.
RISC-V SoC	240.4ms/ Picture	93.6%	This paper
Cortex-A9	7230ms / Picture	95.8%	[10]

6 Conclusion

In the context of mounting demands for enhanced computational performance and power efficiency. This paper developed a basic RISC-V SoC by augmenting the RISC-V architecture with a custom instruction set. This SoC incorporates a singular-launch and three-stage pipeline architecture, in conjunction with a dedicated AI Core computing unit. The integration of a pulsed array serves to address the challenges posed by input/output operations, thereby facilitating the execution of matrix multiplication with optimal efficiency and precise timing.

Performance comparison shows that the design achieves a 30-fold speed increase in convolutional operations compared to conventional CPU serial operations. Experimental findings demonstrate that the processor effectively reduces computational latency and improves overall performance when handling large-scale convolutional operations. The running time of SoC Accelerator is generally increased by nearly 30 times compared with CPU Cortex-A9.

The design's low resource footprint and high power efficiency enable stable operation in resource-constrained edge computing devices, making it particularly suitable for deep learning, image processing, and other applications requiring high

efficiency of convolutional operations. This provides powerful computing support for the development of related fields and has broad application prospects.

References

1. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., Temam, O.: ‘Small Form Factor High Throughput Accelerator for Ubiquitous Machine Learning’, ACM SIGARCH Comp. Archit. News, 2014, 42,(1), pp. 269-28
2. Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., Cong, J.: ' Optimizing FPGA-based deep convolutional neural network gas pedal design'. Proc. ACM/SIGDA Int. Symp. Field - Programmable Gate Arrays, Monterey, California, USA, February 2015, pp. 161–170
3. Kalapothas, S., Galetakis, M., Flamis, G., Plessas, F., Kitsos, P.: ' RISC-V-based Machine Learning Ecosystem Overview', Inform., 2023, 14, (2), pp. 64
4. Bai, L., Zhao, Y., Huang, X.: 'A CNN accelerator on FPGA using depthwise separable convolution', IEEE Trans. Circuits Syst. II, 2018, 65, (10), pp. 1415–1419
5. Kung, H. T.: 'Why systolic architecture', Des. Res. Cent. Carnegie - Mellon Univ., 1982, N/A, N/A, pp. 37–46
6. Qu, Y., Wang, J.: ' FPGA-based low-power YOLO gas pedal design'. Proc. 7th Int. Conf. Advanced Algorithms and Control Engineering (ICAACE), Shanghai, China, March 2024, pp. 1132–1137
7. Sze, V., Chen, Y. H., Yang, T. J., Emer, J. S.: 'Efficient processing of deep neural networks: a tutorial and survey', Proc. IEEE, 2017, 105, (12), pp. 2295–2329
8. Jiang, Z., Zhao, L., Li, S., Jia, Y.: 'Real-time object detection method for embedded devices', Comput. Vis. Pattern Recognit., Seattle, USA, June 2020, Vol. 3, pp. 1–11
9. Chen, S., Lai, W., Ye, J., Ma, Y.: 'A Fast and Low-Power Detection System for the Missing Pin Chip Based on YOLOv4-Tiny Algorithm', Sensors, 2023, 23, (8), pp. 3918
10. Zhang, F., Li, Y., Ye, Z.: ' Applying yolov4-tiny for object detection on an fpga-based convolutional neural network gas pedal', J. Phys. Conf. Ser., Dal, China, May 2022, pp. 012032