Dean&Francis
ISSN 2959-6157

A Highly-Parallel AI Accelerator Architecture for Convolution and Activation, Implemented in Verilog

Qilin Xie[1] **and**

Jingyu Zhang[2, *]

[1]Faculty of Information Science and Engineering, Ocean University of China, Qingdao, 266000, China
[2]Microelectronic Science and Engineering, South China University of Technology, Guangzhou, 510145, China
*Corresponding author: 202364831521@mail.scut.edu.cn

Abstract:

LeNet-5 is a classic Convolutional Neural Network (CNN) model whose core structure, such as the C1 Convolution layer, remains constrained by hardware resources like computing power, power consumption, and storage bandwidth for real-time inference in embedded and edge computing scenarios. To break through this bottleneck and enhance the computing efficiency of artificial intelligence in resource-constrained environments, this study focuses on the design of a dedicated hardware accelerator for the Convolution Layer 1 (C1) of LeNet-5 and its subsequent Rectified Linear Unit (ReLU). A highly parallel convolution computing architecture was constructed, enabling synchronous operation and data reuse across multiple groups of convolution units (CUs), which significantly improved computational throughput and energy efficiency. The experimental results show that while controlling the resource consumption of the Field Programmable Gate Array (FPGA), this accelerator has a significant improvement in inference speed compared with the pure software implementation, successfully verifying the technical feasibility and engineering advantages of hardwareization of the basic operators of Convolution neural networks. This research not only provides a reusable hardware prototype and optimization path for the efficient deployment of CNN models in embedded terminals, but also lays an important theoretical and technical foundation for the industrial application of artificial intelligence edge computing, possessing high academic innovation and practical application value.

Keywords: Convolutional neural networks, FPGA acceleration, hardware implementation, LeNet-5, edge computing

1. Introduction

Against the backdrop of rapid technological development today, embedded systems have been widely applied in numerous fields, such as intelligent home control systems, industry control systems, and intelligent transportation control systems, thanks to their advantages of miniaturization, low power consumption, and high reliability. Real-time computing in embedded scenarios is one of the key technologies driving the intelligent development of the above-mentioned fields. However, the hardware resources of embedded systems are usually limited, which brings many challenges to real-time computing [1]. Among numerous real-time computing tasks, image recognition, as a highly representative and widely applied real-time computing task, can endow embedded devices with visual perception capabilities, enabling them to better perceive and understand the surrounding environment [2].

As an important technology in the field of image recognition, CNN has become a research and application hotspot because of its powerful feature extraction ability and excellent recognition performance. Among them, LeNet-5, as one of the earliest proposed Convolution neural networks, has a simple structure and high computational efficiency, and has achieved remarkable results in the fields of handwritten digit recognition, such as license plate number recognition [3]. Although LeNet-5 is already relatively lightweight compared to other complex CNN models, under the limited resource conditions of embedded scenarios, it still faces huge difficulties in achieving efficient operation of its convolutional layer and activation function layer, and further improving the computing speed of the entire network [4-5].

The main objective of this study is to build a neural network capable of achieving specific image recognition, with a focus on implementing the Convolution layer and activation function layer of LeNet-5 within limited hardware resources and accelerating the Convolution neural network as much as possible. Specifically, the system directly processes the original image input of 32×32×1 to reduce the complexity of preprocessing and enhance the real-time performance of the recognition system. However, the Convolution layer module of Convolution neural networks has the highest computational load and the most repetitions, and it requires a large amount of hardware resources. To achieve this goal, it is necessary to comprehensively balance the resource usage and computing speed. On the one hand, the memory and computing power of embedded systems are limited. Excessive consumption of resources will lead to performance degradation or even failure to operate normally. On the other hand, real-time performance requires the network to have a high computing speed to meet the practical application requirements.

2. The Theory of the C1 Convolutional Layer and Activation Function Layer of LeNet-5

2.1 C1 Convolutional Layer

LeNet-5 is a classic multi-layer artificial neural network architecture. Its main structure combines the Convolution computing and the Spatial downsampling strategy. It has foundational significance in the research of image recognition. This network simulates the biological visual perception mechanism through local receptive fields, significantly reduces the number of parameters by using weight sharing, and achieves spatial dimensionality reduction and translation invariance enhancement of feature maps through downsampling layers [4].

The C1 Convolution layer serves as the first hidden layer and the first feature extraction layer of LeNet-5, directly processing the input image. Its core function is to detect basic visual features such as edges, spots, and changes in light and shade. These features form the basis for subsequent, more complex and abstract features. Layer C1 receives 32×32×1 image data and performs a two-dimensional convolution operation with a step size of 1 and no padding through 6 5×5 convolution kernels, outputting a feature map of 28×28×6 bits [4].

2.2 Activation Function Layer

Activation function layers typically follow convolutional layers (or fully connected layers), applying non-linear transformations to linear outputs. As convolution constitutes a linear operation, without activation functions, the neural network remains equivalent to a linear model regardless of the number of layers, and is incapable of learning complex non-linear patterns. Activation functions endow neural networks with non-linear expressive capabilities, enabling them to approximate arbitrarily complex functions. The classic LeNet-5 network employs the hyperbolic tangent function (tanh) as its activation function.

The original LeNet-5 network employed the tanh function as its activation function, though multiple activation functions are now available for selection. Several activation functions and their advantages are listed below:

The mathematical form of the sigmoid function is based on exponential operations, making it suitable for efficient hardware implementation via lookup tables or CORDIC algorithms. Its bounded output range renders it appropriate for specific network layers requiring probabilistic

ISSN 2959-6157

interpretation or saturation characteristics, such as the output layer.

As a zero-centred activation function, tanh optimises the dynamic range of inputs to subsequent layers, theoretically enhancing training convergence speed. Its hardware implementation resembles a sigmoid, enabling resource reuse through shared exponential units.

ReLU exhibits piecewise linear characteristics, requiring only a comparator and a multiplexer in hardware implementation, resulting in exceptionally concise logic circuits. This low-complexity structure facilitates low latency and high throughput while significantly reducing FPGA resource consumption, such as LUTs and registers.

Following a comprehensive evaluation, ReLU was selected as the optimal solution due to its hardware-friendliness and computational sparsity. Its linear non-saturation characteristic effectively mitigates gradient vanishing issues, thereby accelerating training convergence. In FPGA deployment, the streamlined circuit structure corresponding to ReLU achieves minimal resource consumption and single-clock-cycle processing latency, effectively meeting the hardware pipeline design requirements for high throughput and low power consumption.

Consequently, this study employs the widely adopted modern ReLU function for hardware implementation, rather than retaining the tanh function used in the classic LeNet-5 network.

3. Hardware Implementation of the C1 Convolutional Layer and Activation Function Layer

3.1 Hardware Implementation of the C1 Convolutional Layer

In the C1 Convolution layer, the input image size is 32x32 bits, and the data bit width is float16 (16 bits). Therefore, the input bit width is 32×32×16= 16,384 bits. The size of the convolution kernel is 5×5, with a total of 6 convolution kernels. Each data bit width is float16, so the input bit width of the convolution kernel is 5×5×6×16=2400 bits. The size of the convolution output feature map is calculated by the formula m=[(n-k+2p)/s]+1, where the input size n=32, the convolution kernel size k=5, the step size s=1, and the fill p=0. Therefore, the output size is 28×28 bits, and the output bit width is 28×28×6×16=75,264 bits.

As shown in Fig. 1, the Convolution layer module is constructed and contains 3 convolution calculation modules.

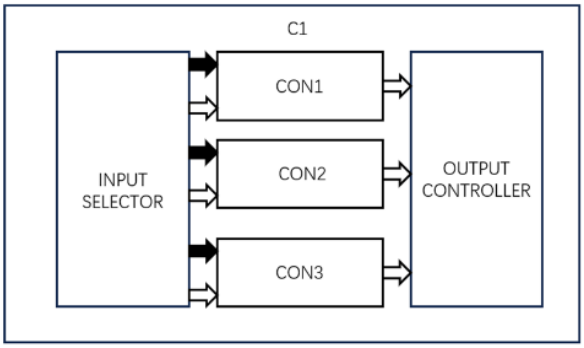

Fig. 1 C1 Convolution layer module

As shown in Fig. 2, it is a construction of the Convolution calculation module.

It consists of 14 CU modules and an input filtering module (Register File, RF), among which the RF module implements the sliding function of the convolution kernel, that is, it controls part of the input allocated to the CU. The input of the CU module consists of the convolution kernel filter and the image covered by the convolution kernel window. The bit width of the filter is 400 bits, the bit width of the image covered by the convolution kernel window is 400 bits, and the output bit width is 16 bits.

The input filtering module (RF) counts through an internal register, similar to a timer. When a batch of convolution kernels (three) has completed convolution, it switches the input to the convolution kernels, enabling the convolution calculation module to complete the calculation of the C1 convolution layer. The CU module has two parts. One is the state control module. Since calculating a 5x5 convolution requires 25 clock cycles (plus 2 cycles of reset delay), sequential processing is adopted to save hardware resources. The other one is that the processingElement16 module contains floating-point multiplication units and floating-point addition units, which implement multiply-accumulate operations and ultimately output convolution results.

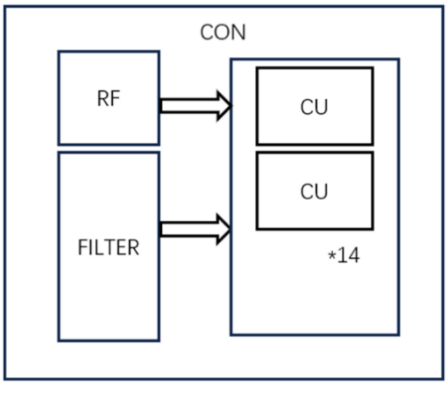

Fig. 2 Convolution calculation module

3.2 Hardware Implementation of the Activation Function Layer

For the 6-channel 28×28 feature map output from the convolutional layer, this design employs a six-channel parallel ReLU activation layer hardware architecture to fully leverage ReLU's low-latency characteristics. Gated clocking technology is introduced to reduce module power consumption. Given the input data is in 16-bit floating-point format, the ReLU implementation requires only sign bit evaluation: if the sign bit is negative, the output is zero; if the sign bit is non-negative, the original input value is directly output. The specific module architecture is illustrated in Fig. 3 below:

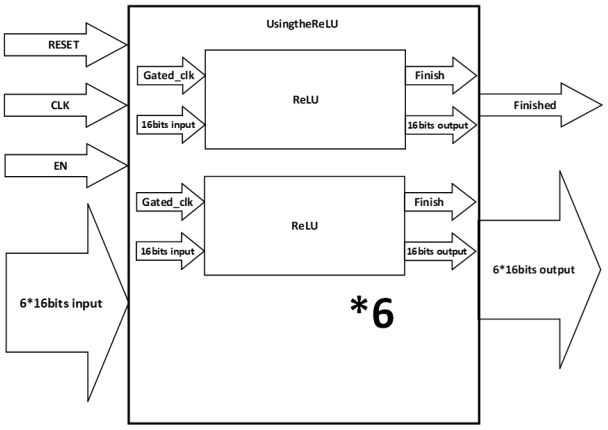

Fig. 3 Activation function layer module

As illustrated in Figure 3, within this architecture, the ReLU computation module is responsible for executing the activation function operation, employing a conditional output strategy based on sign bit determination. Processing each 16-bit input requires only a single clock cycle, achieving single-cycle latency. The host computer integrates six computational modules into a parallel architecture, corresponding to the six-channel output of the convolution operation module. Each clock cycle can receive six-channel 16-bit inputs and generate six-channel 16-bit outputs, enabling highly efficient computation. The gated clock is generated by ANDing an external enable signal with the system clock signal, controlling the operation and power consumption of the internal computational modules. This approach significantly reduces both static and dynamic power consumption at the cost of adding only one gate-level circuit resource. This optimisation proves particularly effective in high-clock-frequency application scenarios.

4. Simulation Results of the C1 Convolutional Layer and Activation Function Layer

4.1 Simulation Results of the C1 Convolutional Layer

This study uses Vivado to synthesize, simulate verification and RTL analysis of the C1 convolution layer. As shown in Fig. 4, it is a simulation result of the Convolution Layer Module.

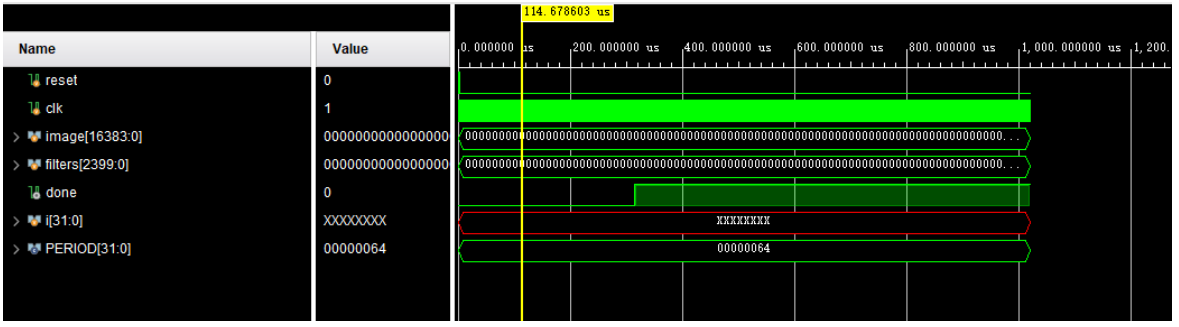

Fig. 4 Simulation waveform output of the convolution module

The schematic diagram generated after the RTL analysis is completed is shown in Fig. 5.

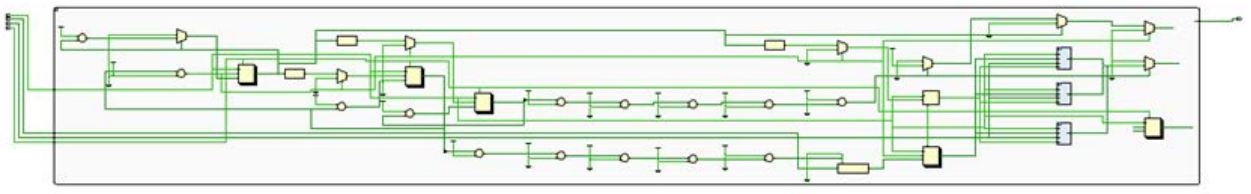

Fig. 5 Schematic Diagram of the convolutional module RTL

ISSN 2959-6157

Table 1. Resource Utilization of the convolutional module

Resource	Utilization
LUT	621585
FF	117655
DSP	44

According to Table 1, it can be concluded that this module has a high level of parallelism and fast computing speed, but the resource consumption is relatively large. The LUT uses 621585. Although the high degree of parallelism design optimizes the computing speed, the resource consumption is very large. If the entire LeNet5 convolutional neural network is to be installed, it is necessary to perform time-division multiplexing on the control module and the computing module. Only in this way can the LeNet5 neural network be successfully installed on the FPGA.

4.2 Simulation Results of the Activation Function Layer

Within the Vivado Integrated Development Environment, the activation function module in this design underwent synthesis, simulation, and RTL analysis. Simulation testing covered multiple input scenarios, including all-zero inputs, mixed positive and negative values, identical values, and dynamic control via the enable signal. In all scenarios, the module behaved as intended. When the enable signal was active, it consumed only one clock cycle to output the correct result for valid inputs and set the completion flag to active. When the enable bit was inactive, the module suspended operation and waited for the enable bit to become active again, thereby reducing both static and dynamic power consumption. Functional correctness was comprehensively verified. Specific simulation waveforms are shown in Fig. 6:

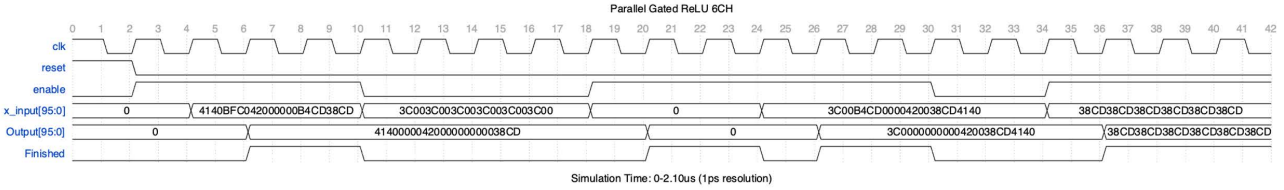

Fig. 6 Simulation waveform output of the activation function module

The RTL analysis phase generated the structural schematic diagram for this module, with results shown in Fig. 7. This illustrates its synthesised logic composition and signal interconnections. When the module enable signal is active, the six-channel parallel processing module generates six parallel output results one clock cycle after input arrival. It simultaneously integrates and outputs the completion signals from all six processing modules. Concurrently, it detects new inputs in real time, enabling automatic reset of the completion signals. When the module is disabled, it enters a dormant state. The processing completion signal remains inactive, and the module neither responds to nor processes inputs. The output persists at its original value.

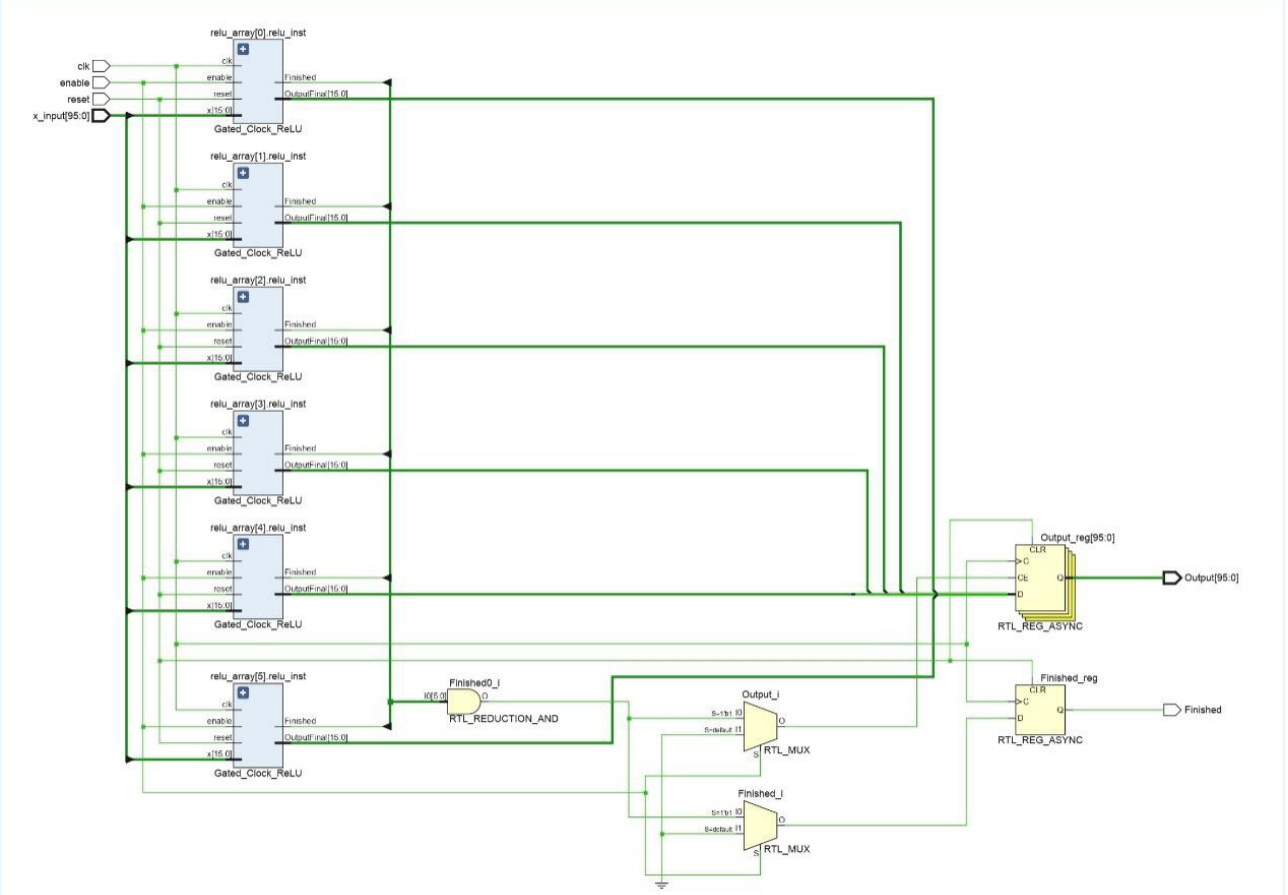

Fig. 7 Schematic diagram of the activation function module RTL

The resource utilisation report for the module on the target FPGA device is presented in Table 2. The data indicates that this activation function module occupies only 52 LUTs, 183 FFs, and 196 I/O pins, achieving the intended computational objectives with minimal logic resource consumption.

Table 2. Resource Utilization of the activation function module

Resource	Utilization
LUT	49
FF	92

In summary, this activation function module features a streamlined design and efficient hardware implementation, exerting minimal resource overhead on the overall system. Its lightweight nature renders it well-suited for deployment within FPGAs as an auxiliary processing unit, capable of working in high synergy with high-load modules such as convolutional computation units.

5. Discussion

This study presents the implementation of the C1 layer of a convolutional neural network accelerator using a manual RTL-based design methodology. At the convolution computation level, compared to automated approaches based on high-level synthesis, parameterised configurable architectures, and resource-folding architectures, this design employs customised fixed-size computational unit arrays and a fully unfolded parallel architecture [5-7]. This enables fine-grained control over computational units and data pathways, demonstrating excellent peak performance potential.

In terms of activation function design, this approach employs the ReLU function coupled with gated clocking technology, yielding an order-of-magnitude advantage in energy efficiency over traditional tanh/sigmoid implementations [8]. Compared to LeakyReLU, it significantly re-

ISSN 2959-6157

duces circuit complexity through simplified decision logic [6]. Compared to activation function implementations in general-purpose frameworks, this design achieves synergistic optimisation through simplified logic structures and gated clocking technology. This ensures functional completeness while substantially reducing resource overhead and power consumption [9]. Furthermore, it employs a flow control mechanism to manage states by automatically detecting input variations. This approach incurs lower resource overhead than complex scheduling reliant on global central controllers, requires no external intervention, and significantly simplifies system control logic [9]. This architectural innovation offers novel design insights for accelerating neural networks in edge computing scenarios. Building upon this foundation, the technical characteristics of this design confer significant application value across multiple edge computing scenarios. Its highly parallel architecture is particularly well-suited for high-resolution real-time video analysis systems, serving as an efficient front-end feature extraction engine for processing 1080p and even 4K video streams. In power-sensitive wearables and IoT terminals, optimised computational units and gated clocking techniques substantially enhance device endurance. Simultaneously, as a validated high-performance computing core, this design serves as foundational IP for constructing heterogeneous computing systems, delivering reliable underlying computational support for specialised domains such as autonomous driving and industrial quality inspection.

Nevertheless, it must be candidly acknowledged that this research design exhibits several limitations. Firstly, to achieve extreme optimisation of single-layer computational performance, this design incurs substantial hardware resource costs—a single convolutional layer consumes over 620,000 LUTs, posing significant challenges for practical deployment on current mainstream edge computing platforms. Secondly, as an accelerator design focused solely on single-layer optimisation, this solution has yet to establish a complete system-level implementation, requiring further refinement in inter-layer data integration and control scheduling mechanisms. Moreover, the design employs the float16 floating-point format without incorporating fixed-point quantisation techniques, leaving room for improvement in both storage and computational efficiency. These technical shortcomings collectively constrain the transition of this design from theoretical research to practical application, while also indicating key directions requiring breakthroughs in subsequent research.

Based on this, this paper posit that the future evolution of convolutional neural network hardware accelerators will witness three prominent trends. First, specialization and generalization will undergo deep integration. The specific operator optimizations demonstrated in this design can be encapsulated into reusable IP cores, effectively complementing programmable general-purpose architectures. This trend is exemplified by emerging heterogeneous computing systems that leverage FPGAs alongside CPUs and GPUs, highlighting the synergistic cooperation between custom computing devices and general-purpose processors [10]. Second, algorithm-hardware co-design will advance toward finer granularity. The future will see a rise in dedicated hardware units tailored for novel operators, facilitated by intelligent compilers that enable efficient and reliable automated mapping [11]. Finally, hierarchical co-optimisation of storage and computation will become pivotal for overcoming the memory wall bottleneck. This relies on more intelligent data scheduling strategies and compute architectures that operate close to storage, particularly the continued development and maturation of near-memory computing technologies [12]. Together, these technical pathways will shift the focus of neural network accelerators from isolated computation acceleration to holistic system-level performance optimization, thereby establishing a more robust and dependable hardware foundation for edge intelligence.

6. Conclusion

This study implements specialised accelerators for the C1 convolution layer and ReLU activation layer within the LeNet-5 network using Verilog. The design employs a parallel computing architecture and gated clocking technology, achieving outstanding energy efficiency while maintaining numerical precision. Hardware simulation verifies its comprehensive advantages in computational throughput and power consumption control, providing a reliable hardware solution for edge-side intelligent computing.

Innovations in specialised neural network accelerators are driving the evolution of computational paradigms from general-purpose architectures towards domain-specific customisation. This shift will accelerate the widespread adoption of emerging technologies such as 3D integration and compute-in-memory within the industry. With the deep integration of algorithms like sparse computing and dynamic precision into hardware, FPGA-based heterogeneous computing architectures are emerging as the core of next-generation intelligent computing platforms. These technological breakthroughs will drive comprehensive restructuring of computing systems from chip-level to system-level, overcoming traditional energy efficiency bottlenecks through hardware-software co-design. This paves new technical pathways and development opportunities for the entire computing industry.

Authors Contribution
All the authors contributed equally and their names were listed in alphabetical order.

References

[1] Liu Z, Dou Y, Jiang J, et al. Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2017, 10(3): 1-23.
[2] Li H, Cheng C, Juntong Y, et al. Multi-Scale Feature Fusion Convolutional Neural Network for Indoor Small Target Detection. Frontiers in Neurorobotics, 2022, 16881021-881021.
[3] Zhao Z, Yang S, Ma Z. Research on Vehicle Licence Plate Character Recognition Based on the Convolutional Neural Network LeNet-5. Journal of System Simulation, 2010, 22(03): 638-641.
[4] Yu N, Jiao P, Zheng Y. Handwritten digits recognition based on improved LeNet5//The 27th Chinese control and decision conference (2015 CCDC). IEEE, 2015: 4871-4875.
[5] Wang Y, Xie K, Chen S, Hu J, Chang S. A universal design on hardware acceleration of convolutional neural networks. Computer Engineering & Science, 2023, 45(4): 577-581.
[6] Zhou S, Qian S, Wei S, et al. Low-Power Neural Network Accelerator for Edge Deployment. Automation and Instrumentation, 2024, 39(07): 147-151+156.
[7] Deng Y. Research on FPGA-Based Convolutional Neural Network Accelerators. Changchun University of Technology, 2025.
[8] Pang M, Wei X, Zhang Y, et al. Efficient Adaptive Convolutional Neural Network Accelerator for Low-Resource Chips. Computer Science, 2025, 52(04): 94-100.
[9] Basalama S, Sohrabizadeh A, Wang J, et al. FlexCNN: An end-to-end framework for composing CNN accelerators on FPGA. ACM Transactions on Reconfigurable Technology and Systems, 2023, 16(2): 1-32.
[10] Samayoa W F, Crespo M L, Cicuttin A, et al. A survey on FPGA-based heterogeneous clusters architectures. IEEE Access, 2023, 11: 67679-67706.
[11] Cong J, Lau J, Liu G, et al. FPGA HLS today: successes, challenges, and opportunities. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2022, 15(4): 1-42.
[12] Singh G, Alser M, Cali D S, et al. FPGA-based near-memory acceleration of modern data-intensive applications. IEEE Micro, 2021, 41(4): 39-48.