A High Performance Vision Transformer Accelerator

Yu Shi

***Abstract*—. Transformer models based on attention mechanisms have shown superior performance in the field of computer vision. Designing dedicated accelerators for Transformers can significantly enhance inference performance and reduce power consumption. In this paper, we first employ an integer quantization strategy for both features and weights to reduce the storage requirements and computational complexity of the model. Then, based on the computational characteristics of convolution and attention mechanisms, we design an efficient and flexible hardware architecture, including a multidimensional systolic array and nonlinear normalization acceleration units. This architecture efficiently maps algorithms to the systolic array, optimizing data storage efficiency and minimizing data movement. Additionally, the accelerator design is implemented on an FPGA. Experimental results show that the proposed FPGA accelerator improves model inference speed with minimal accuracy loss. The ViT-base model achieves an average throughput of 626.2 GOPS on this accelerator, with a synthesized power consumption of 16.4W at a 200MHz clock frequency.**

***Index Terms*—FPGA, Systolic Array, Transformer, Hardware Accelerator**

I. INTRODUCTION

In recent years, Transformer networks based on attention mechanisms [1][2][3] have demonstrated superior performance compared to traditional Convolutional Neural Networks (CNNs) [4][5], primarily due to their ability to model long-range dependencies and allow parallelization across input sequences. This has made them a hot research topic in the field of computer vision. For certain applications, such as autonomous driving and drone navigation, computer vision tasks require real-time performance on edge devices. This demand has driven the development of energy-efficient hardware accelerators [6][7] for inference based on traditional CNN models. Due to their high parallelism, low latency, and low power consumption, Field-Programmable Gate Arrays (FPGAs) have demonstrated higher energy efficiency compared to Graphics Processing Units (GPUs) and Central Processing Units (CPUs), and are therefore widely used for accelerating deep learning algorithms [8][9].

One of the primary challenges in accelerating ViT on hardware platforms is the enormous number of parameters in Transformer models. The attention mechanism in Transformer networks involves multiple large matrix multiplications and data interactions, which results in high computational and storage overhead. This makes it difficult for some embedded devices and mobile terminals to directly store such large parameter sizes. As attention-based Transformer models continue to scale up, the computational load also increases, with ViT-H/14 having a parameter size of 632MB [10]. Additionally, Transformer networks heavily rely on complex nonlinear operations such as LayerNorm and Softmax, which are embedded into the ViT architecture to improve performance [11]. For instance, LayerNorm is applied after each self-attention mechanism or Multi-Layer Perceptron (MLP) sub-layer, normalizing the output of each layer to have zero mean and unit variance. The Softmax function uses the natural exponent as a nonlinear function to amplify small differences in input values, thereby increasing classification accuracy. These nonlinear operations significantly increase the computational load due to floating-point arithmetic. One potential solution is integer quantization. Li et al.[12] proposed a mixed-precision quantization method for vision transformers, combining fixed-point and exponential quantization. By modeling the total computational resources and target frames per second (FPS) of the FPGA, the optimal ratio between fixed-point and exponential quantization was determined. However, this mixed-precision quantization approach is challenging to train. In contrast, Li et al.[13] introduced an integer-only quantization model for ViT, enabling inference entirely with integer arithmetic and bit-shifting. Nevertheless, this design is based on GPU platforms, and the quantization scheme still requires a significant number of division operations, which may be unsuitable for resource-constrained edge devices.

To address the above challenges, this paper proposes a high-performance ViT accelerator. The main contributions of this paper are as follows:

Contribution 1: We analyze the data flow of ViT and extract key operations such as convolution, matrix multiplication, and nonlinear functions. Based on the computational characteristics of attention mechanisms and convolutions, we design a multidimensional systolic array architecture. This architecture optimizes data selection and arrangement, supports convolutions and matrix multiplications of various sizes, significantly enhances the computation density and energy efficiency of the accelerator, and reduces system power consumption.

Contribution 2: Through an integer quantization strategy, we compress the parameter size of the ViT model to one-quarter of the original size in image classification tasks, while controlling the accuracy loss to within 2.4%. Additionally, we split the fixed-point computation of nonlinear operations like Softmax and LayerNorm into multiple stages and designed hardware-friendly nonlinear computing units, thereby significantly reducing memory consumption and computational costs.

Contribution 3: Based on the algorithm and hardware-level optimizations proposed above, we implemented the hardware

Manuscript received March 20, 2025

Yu Shi, School of Software, Tiangong University, Tianjin, 300387, China

accelerator on the Xilinx FPGA platform. The ViT-base model achieved an average throughput of 626.2 GOPS on this accelerator. Compared to existing FPGA-based accelerators, the average throughput increased by 1.6x to 3.3x. Compared to CPU and GPU implementations, energy efficiency improved by 7.8x and 1.6x, respectively.

II. Procedure for Paper Submission

A. Vision Transformer

The Vision Transformer (ViT) is the first Transformer-based model proposed for image classification tasks, and it has demonstrated performance superior to CNNs on large-scale datasets[10]. ViT is composed of three modules: the embedding layer, the Transformer encoder, and the MLP Head. The embedding layer is designed to convert the input three-dimensional image data into a one-dimensional token sequence required by the Transformer. The Transformer encoder is responsible for extracting global features from the input image and is formed by stacking L identical layers (L = 12 in ViT). Each layer consists of two sub-layers: a multi-head attention layer and a multi-layer perceptron (MLP).

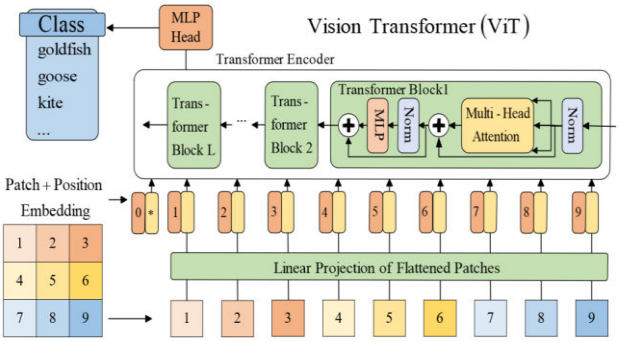

Figure1 Architecture of the Vision Transformer network.

Specifically, after feeding the class embedding vector into the encoder, the Transformer encoder achieves interaction or aggregation of image features through the self-attention layer and then passes the class token to the MLP Head for classification prediction. The MLP Head is responsible for the final multi-class classification task. It takes the global features as input and computes the classification prediction by using the class embedding vector from the encoder and the image label loss, which backpropagates to constrain the network during training.

B. Quantization of Vision Transformer

Quantization is a hardware-friendly model optimization technique in deep learning that aims to reduce storage and computation resource consumption by converting high-precision data into low-precision data, thereby improving computational efficiency on specialized hardware. Uniform Quantization[14] is widely applied on hardware platforms. In this design, the Conv, Linear, and MatMul modules all adopt a uniform MinMax quantization strategy. As deep learning models continue to expand, particularly in large-scale data processing and complex tasks, computational overhead and energy consumption have become major bottlenecks in real-world deployment. The introduction of quantization techniques effectively alleviates these issues, enabling models to run efficiently on resource-constrained devices such as mobile and embedded systems, while significantly reducing storage requirements and computational costs when deployed in the cloud.

The challenge of quantization lies in balancing accuracy and performance. To preserve model performance as much as possible, various techniques are employed during the quantization process, such as quantization-aware training (QAT). This approach allows the model to account for the impact of quantization on accuracy during training and make necessary adjustments. As a result, despite using lower precision after quantization, the model can maintain performance close to that of the original floating-point model in most application scenarios.

FQ-ViT [15] fully quantizes the ViT model and introduces the Power-of-Two Factor (PTF) to handle severe inter-channel variations in the inputs to LayerNorm. Additionally, it proposes an integrated quantization solution, Log-Int-Softmax (LIS), to achieve 4-bit quantization of the attention map. During inference, BitShift operators are used in place of matrix multiplication (MatMul), effectively reducing hardware resource requirements. PoT[12] leverages weights and activations represented as powers of two, typically using 2-4 bits per value, leading to smaller memory footprints and faster computation. However, the accuracy loss from PoT quantization is greater than that from FQ-ViT quantization. For example, the benchmark for FP32 DeiT-s shows up to 79.85% TOP-1 accuracy. With FQ-ViT quantization, accuracy drops to 78.4%, while PoT quantization results in a 77.97% accuracy. In this design, both LayerNorm and Softmax adopt the FQ-ViT quantization scheme.

The quantization process primarily consists of the following three steps:

(1) Dynamic Range Mapping: In symmetric quantization, the absolute maximum value of the input data is typically used to determine the dynamic range. In asymmetric quantization, the dynamic range is usually calculated as the difference between the minimum and maximum values. The purpose of computing the dynamic range is to determine the number of bits used for quantization. A linear method is then applied to map the floating-point value x to the lower-precision target range, as expressed in Equation(1).

$$Q(x) = \text{round}\left(\frac{x-z}{s}\right) \quad (1)$$

(2)Quantized Computation: Using the previously defined formula, the quantized value (Q(x)) is computed to discretize floating-point numbers, converting weights and activation values into low-precision integers.

(3)Dequantization: During the inference phase, the quantized values $Q(x)$ are restored to floating-point values using a fixed scaling factor to complete necessary computations, as described in Equation (2).

$$x = s \cdot Q(x) + z \quad (2)$$

C. Accelerators of Transformer

Before designing accelerators, pre-processing operations such as pruning and compression are typically applied to the network model to reduce the number of parameters, simplify computational complexity, and lower hardware resource consumption. Liu et al. [16] proposed Fully Quantized BERT

(FQ-BERT) to introduce variable bit-widths. Since different layers of BERT have varying bit-widths, a reconfigurable module design with variable bit-widths was adopted. They presented a bit-level reconfigurable multiplier accelerator, which supports 8/4-bit and 8/8-bit multiplications. Wang et al. [17] designed an accelerator called SpAtten to co-process the self-attention layers in NLP tasks. This accelerator supports a novel token pruning technique to reduce memory access and computational load in self-attention layers. The entire accelerator module is fully pipelined, with each module corresponding to a specific operation, thereby minimizing data movement and greatly reducing the overhead associated with data transfers.

III. HPVIA ACCELERATOR DESIGN

A. *Hardware Accelerator Architecture*

Based on the modular partitioning of partial mapping schemes, the overall architecture of the hardware accelerator can be further designed, as shown in Figure 2. To efficiently allocate computing tasks, optimize data flow, and enhance hardware utilization, the hardware architecture incorporates a 2D systolic array cluster, nonlinear computation units, and a three-tier memory hierarchy.

The three-tier memory hierarchy consists of Off-Chip Memory, On-Chip Memory (SRAM), and Local Registers (Reg). As the primary storage unit for large-scale model parameters and input data, Off-Chip Memory is connected to On-Chip Memory (SRAM) via a high-speed data bus to ensure fast data access. In each attention mechanism computation, the local registers (Reg) buffer intermediate results to reduce repeated access to SRAM, thereby lowering overall power consumption. The Q matrix and K matrix obtained from the input module are stored in the left and right registers, respectively. The intermediate Attention matrix is stored in the Softmax register and processed by the Softmax unit. Afterward, the Q matrix in the left register is refreshed and multiplied with the V matrix stored in the right register. Once an MSA (Multi-Head Self-Attention) computation is completed, the results are quantized to 16-bit precision and written back to SRAM0.

For matrix multiplication in MLP (Multi-Layer Perceptron) computations, the left and right inputs of the systolic array come from SRAM1 and Parameter SRAM, respectively. The block-wise output of FC1 (first fully connected layer) undergoes GELU activation, is quantized to 16-bit precision to reduce computational and storage costs, and is written back to SRAM1. The FC2 (second fully connected layer) computation results are also quantized to 16-bit and stored in SRAM0.

The Off-Chip Memory communicates with SRAM0 via the Data Bus, reading results and writing the next set of inputs for subsequent computations. Meanwhile, MSA and MLP parameters are stored in Parameter SRAM using an alternating buffering strategy to reduce data transfer latency.

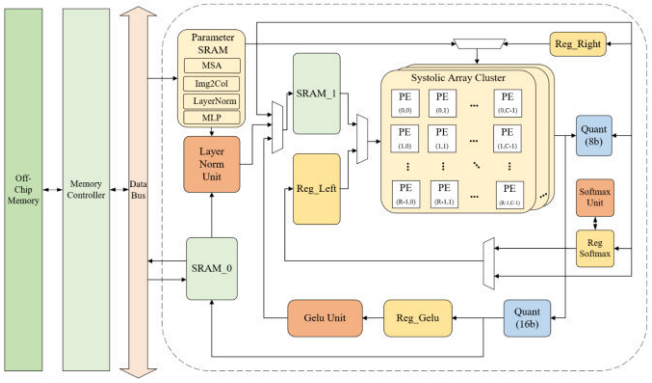

Figure2 Architecture of Hardware Accelerator.

B. *Data Layout Scheme*

FPGA-based convolution acceleration schemes typically achieve efficiency by parallelizing the core convolution operations, such as multiplication and addition, thereby directly executing convolution calculations without relying on additional algorithms for transforming convolution into matrix multiplications. However, one major limitation of this approach is its inability to flexibly support convolution operations of varying kernel sizes. Consequently, each distinct convolution kernel size necessitates specific hardware logic development. To maximize the reuse of FPGA logic and memory resources, this research proposes mapping convolution operations into an Im2Col representation combined with a General Matrix Multiplication (GEMM) layout. Through this approach, all types of general matrix multiplication computations can be executed using identical computation kernels and on-chip memory structures, substantially improving hardware resource utilization and computational efficiency.

Figure3 illustrates the Im2Col + GEMM data layout, where 'N' denotes the batch size, [H,W] represents the output feature map's height and width, [K,K] signifies the filter dimensions of the current convolutional layer, C_{in} is the channel depth of the current convolutional layer, and O_k denotes the output channel depth. In this scheme, the Feature Map Matrix A is obtained through the Im2Col operation based on the parameters of the neural network under consideration. Depending on the tensor layout, the order of elements within each row of matrix A may vary. Assuming an element ordering of [N, H, W, C_{in}] in matrix A, the convolution kernel weights and each sliding window are unfolded along the channel dimension. Consequently, each row in matrix A corresponds to one batch of the input feature map, sequentially flattened in the order [H, W, C_{in}].

The columns of the Weight Map Matrix B represent convolutional weights for each output channel, flattened sequentially following the order [K, K, C_{in}]. Meanwhile, the rows of Output Feature Map Matrix C correspond to all channels of each pixel within the output feature map. Taking the embedding layer of a ViT-base model as an example, a convolution kernel with dimensions [K, K, C_{in}, O_k] = [16, 16, 3, 768]' is employed. With a stride of 16, the input feature map of size (H, W) = (224, 224) is segmented into sub-images of size '(14, 14)'. After flattening, each sub-image is treated as a token input into the encoder, wherein each token corresponds to a distinct row in matrix C.

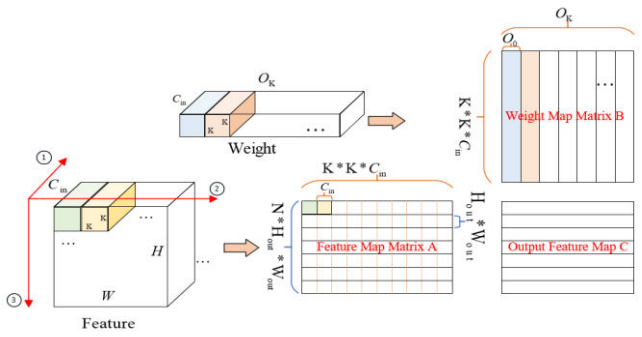

Figure3 Layout of the Im2Col + GEMM.

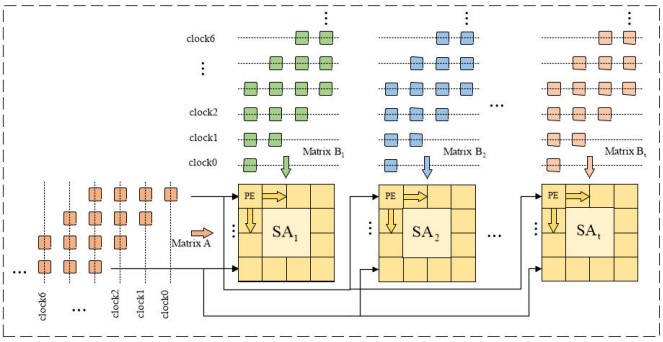

Figure4 Architecture of the Systolic Array Cluster.

C. *Systolic Array Cluster*

To enhance the throughput and energy efficiency of the systolic array, this paper proposes increasing the number of systolic array clusters to elevate parallelism levels. Existing systolic array architectures can thus be scaled to form a cluster-based systolic array, as $[T, H_{SA}, W_{SA}]$. Every H_{SA} rows of matrix A will progressively interact with $T \times W_{SA}$ columns from matrix B, producing a resultant matrix of size $[H_{SA}, T \times W_{SA}]$.

Taking the ViT-base model as an example, we consider the calculation of multi-head self-attention (MHA) using a two-dimensional systolic array cluster. The input activation tensor is sized [197, 768], and the weight matrices W_Q, W_K, and W_V each have dimensions of [768,768]. After performing matrix multiplications, three Query-Key-Value matrices, each sized [197,768], are obtained.

In the multi-head self-attention mechanism, these matrices are reshaped based on the number of attention heads. Assuming a total of 12 attention heads, each Q-K-V matrix is reshaped into a three-dimensional tensor with dimensions [12,197,64]. The context information is calculated by performing $Q \times K^T$ where Q represents the Query matrix and K^T the transposed Key matrix. Given that Q, K, and V matrices are high-dimensional tensors, they must be segmented into smaller submatrices. This segmentation introduces data discontinuity, complicating deployment on edge devices. Furthermore, when multiple attention heads are employed, additional data movements such as reshaping, transposing, and concatenation operations are required, resulting in extra time overhead.

To mitigate this issue, we observe that the systolic array inherently outputs data in a column-first manner, traversing from left to right and top to bottom. Consequently, when data are fed into the systolic array in a fixed direction, the transpose operation on the Key matrix K can be eliminated. Leveraging the predictability of linear layer weights, we propose rearranging these weights at compile time to match the systolic array's computation requirements. These reordered weights, combined with outputs from the Embedding layer, enable efficient computation of the multi-head self-attention mechanism.

D. *Hardware design of LayerNorm*

In the ViT model, Layer Normalization (LN) normalizes the outputs of each layer to have zero mean and unit variance, which is performed individually for each layer rather than across batches. This normalization method contributes to a more rapid and stable training process and can be expressed as follows when combined with the PTF quantization method:

$$\text{LayerNorm}(X) = \frac{(WX_Q - M_1)}{\sqrt{WM_2 - M_1^2}} \times \gamma + \beta \tag{3}$$

E. *Hardware Design of LayerNorm*

Unlike BatchNorm, which obtains fixed parameters during training and allows preprocessing during inference, LayerNorm (LN) requires dynamic computation of statistical parameters. This involves an initial pass over all input data to determine the mean before proceeding with further calculations, making it highly hardware-unfriendly.

To address this, the integer-based LN computation is divided into two stages, as illustrated in Figure 5.

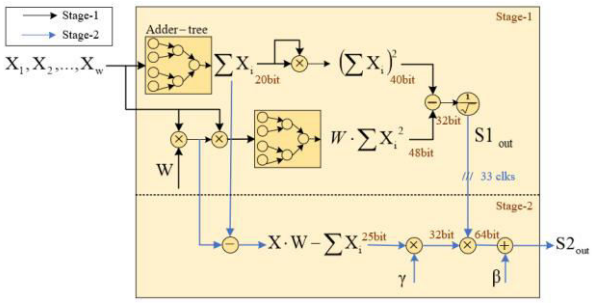

Figure5 Architecture of the Interger-based LayerNorm.

F. *Hardware Design of Softmax*

In multi-head self-attention (MHSA), the Transformer model computes multiple attention heads in parallel to capture different semantics and dependencies in the input sequence. As a crucial component of MHSA, Softmax converts attention scores into probabilities, ensuring the model's sensitivity to different inputs. Softmax accounts for a significant portion of the total runtime of the Transformer network.

The computation of exponential and logarithmic functions is often a performance bottleneck. However, it can be approximated using linear function shifts, multiplications, and additions, reducing the reliance on complex mathematical operations. This approach effectively utilizes

hardware resources, enabling accelerated computation and lower power consumption.

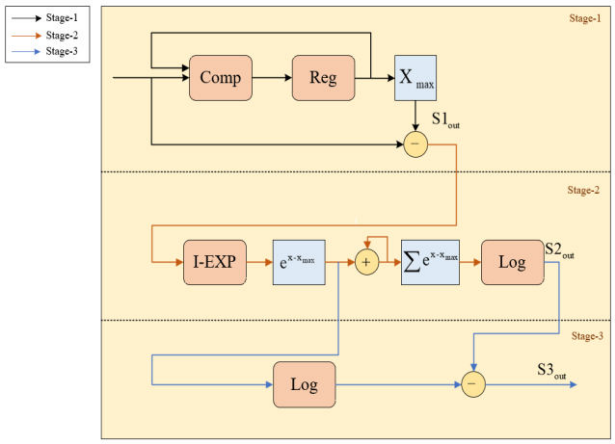

Figure6 Architecture of the Interger-based Softmax.

IV. EXPERIMENT

The experimental environment consists of two parts: the host and the FPGA, with specific configurations shown in Table 5-1. The host is equipped with an i9-12900H CPU for running software programs of the hardware accelerator and an NVIDIA Tesla V100 GPU server for training the ViT model. The server software environment includes Python 3.6.10, PyTorch 1.4.0, GCC 7.3.0, and CUDA 10.1.

On the FPGA side, the Vivado 2022.2 development suite is used for logic simulation and bitstream generation. The accelerator is implemented in C and compiled in the GCC 7.3.0 environment. The Vivado HLS development tool is used to convert C code, along with pipeline and parallel optimization instructions, into functionally equivalent Verilog code to realize the hardware accelerator's functionality. After exporting the hardware IP core, the IP-based peripheral circuit design is completed in Vivado, and the bitstream is generated. Finally, system-level testing and verification are conducted on the board.

Through software-hardware co-design, the hardware accelerator in the PL (Programmable Logic) side directly interacts with the processor system via the AXI protocol. The PS (Processing System) side configures and schedules the accelerator by sequentially reading toolchain-generated instructions, enabling efficient task execution and resource management. Instructions are first tested in software before being transmitted and stored in on-chip memory, while input data and model parameters are stored in off-chip memory.

After designing the accelerator, a testbench file needs to be written for on-board verification to ensure functional accuracy. This board integrates an ARM core (PS side) and FPGA logic units (PL side), providing hardware resources such as 274K LUTs, 548K FFs, 2520 DSP blocks, and 912 on-chip Block RAMs, with an operating frequency of up to 200 MHz.

During system operation, the ARM core acts as the software control unit within the hardware architecture. It first stores input data and model parameters in off-chip DDR memory and loads instructions into on-chip BRAM memory. During execution, the ARM core controls data and parameter transfers through interrupt requests, coordinating hardware computation units to complete computation tasks.

In software, models typically use float16 or float32 for parameters and input feature images. However, on FPGA reconfigurable devices with limited resources, as the data volume increases, excessively high numerical precision may lead to reduced computational efficiency and increased inference latency, thereby limiting the parameter scale of deployable models.

To address this issue, this study adopts the hardware-friendly quantization strategy FQ-ViT and inserts quantization nodes between each ViT operator to monitor quantization errors and accuracy loss. The experiment trains and quantizes three ViT networks—ViT-Base, ViT-Small, and ViT-Tiny—on the ImageNet 2012 dataset. The quantization method effectively reduces the computational burden and storage demand of ViT on FPGA and other hardware platforms while minimizing the impact of quantization on model accuracy. This improves both the computational efficiency and resource utilization of ViT deployment on FPGA. The error statistics between the quantized lightweight model and the full-precision floating-point model are shown in Figure 7.

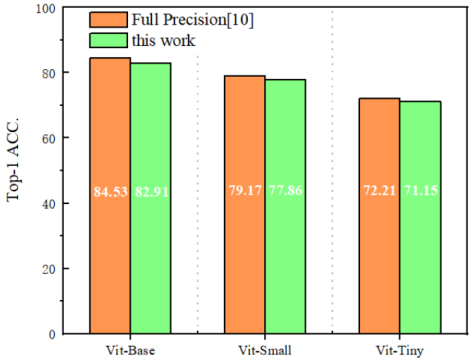

Figure7 Result of Quantization.

Due to the unavoidable rounding operations during quantization, model accuracy decreases by 1.62%, 1.31%, and 1.06%, respectively, but remains within an acceptable range. Additionally, using INT8 computation significantly reduces on-chip resource usage. On FPGA devices, INT8 storage resources require only 1/4 of the storage needed for FP32, and the hardware resources required for fixed-point 8-bit computations are far lower than those for 32-bit floating-point computations. This ensures high computational efficiency while significantly improving hardware resource utilization.

After determining the size of the systolic array, the hardware resource utilization of the development board, as shown in Table 1, was analyzed.

First, the systolic array, responsible for intensive matrix computation tasks, accounts for 92.4% of the total DSP usage. This proportion aligns with the computational requirements of ViT hardware acceleration, indicating that the computation load is primarily concentrated in the matrix operation module. This proportion is consistent with ViT's need for large-scale matrix computations, ensuring high inference efficiency for the ViT model on FPGA. The GEMM, Im2col, Transpose, and Bit-Width Conversion

modules are primarily used for data rearrangement, format conversion, and access, functioning as data flow optimization components. These modules rely mainly on look-up tables (LUTs) and flip-flops (FFs) for data block rearrangement and access, without consuming DSP resources, thereby effectively reducing computation resource usage. The remaining hardware resources, after accounting for the systolic array, are mainly allocated to the non-linear operations Softmax, LayerNorm, and GELU. These modules involve exponential calculations, normalization, and activation functions, which typically require a certain amount of DSP resources. However, compared to matrix operations, their computational burden is relatively small. Finally, the AXI and DDR-related logic is mainly used for optimizing data transmission, ensuring efficient interaction between the accelerator and off-chip memory. These components additionally occupy 4.3% of the total LUT usage and 1% of FF usage, serving storage control and data management functions to ensure efficient data flow and maintain high throughput for ViT inference tasks.

Table1 Resource utilization

ZCU102	LUT	FF	BRAM	DSP
Systolic Array	149977	145524	0	2048
GEMM	1200	877	16	0
Im2Col	1664	1630	38.5	0
On-chip Memory	2116	870	512	0
Transpose	953	838	8	0
Softmax	1834	1735	8	46
LayerNorm	1716	1682	6	43
GELU	10163	5992	0	32
Concat&Add	1019	1878	8	8
Bit-With Conversion	770	128	8	0
AXI and DDR related	7825	16403	15	40
Total	182743	177557	637	2217

To comprehensively evaluate the performance of the proposed accelerator, we decomposed the entire neural network task and tested the accelerator's performance in processing each network layer. The entire ViT neural network consists of three parts: an embedding layer that partitions image input data, a Transformer encoder layer that extracts global feature information, and a classification layer that outputs classification results. Taking the ViT-base model as an example, this model includes 12 Transformer modules, each with 12 attention heads, a patch size of 16, 196 patches, a patch depth of 768, and an encoder dimension of 768. Table 5-2 summarizes detailed information about the ViT-base model, including input and output tensor data shapes, throughput, and acceleration latency for each operation.

As shown in Table 2, the total latency for the entire network is only 53.7 milliseconds, with an average throughput of 626.2 GOP/s. The overall power consumption of the accelerated ViT network is approximately 16.4W, offering a superior energy efficiency ratio compared to GPUs, making it suitable for high-efficiency computing scenarios. The most time-consuming operations are convolution (Convolution) and matrix multiplication (Linear), both of which involve extensive linear transformations and feature computations, requiring significant computational resources.

Table2 Performance of accelerator

	Operation		Delay (ms)	Throughput (GOP/s)
Embed	Convolution		0.946	684.4
	Concat		0	-
	LayerNorm		0.107	-
Encoder ×12	MSA	Linear	0.976	714.5
		$Q \times K^T$	0.201	312.9
		Softmax	0.028	-
		$V^T \times Attn^T$	0.22	465.7
		Add	0.096	-
	MLP	LayerNorm	0.108	-
		Linear+Gelu	1.355	653.3
		Linear+Add	1.339	693.9
MLP	LayerNorm		0.108	-
	Full Connection		0.061	26.68
Total	-	-	53.7	626.2

V. Conclusion

In this paper, we propose a high-performance FPGA-based ViT accelerator architecture. By adopting INT8 quantization and co-designing the hardware and software, we enable the execution of nonlinear operations and other inference processes in ViT using integer arithmetic or shift operations. Additionally, we developed a unified data encapsulation strategy and optimized both on-chip and off-chip data storage and transfer strategies to minimize memory access time and enhance overall efficiency. We also designed a configurable multi-dimensional systolic array to execute various matrix multiplication operations with high parallelism. This acceleration scheme has been implemented on the Xilinx ZCU102 FPGA platform and applied to accelerate three typical ViT network models: ViT-base, ViT-small, and ViT-tiny. Experimental results show that the proposed accelerator achieves an average throughput of 626.2 GOPS on the ViT-base model, representing a 1.8x to 3.7x improvement in average throughput compared to existing FPGA accelerators. Moreover, compared to traditional CPU and GPU implementations, the energy efficiency is improved by 8x and 1.7x, respectively.

References

[1] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[2] Radford A, Kim J W, Hallacy C, et al. Learning transferable visual models from natural language supervision[C]//International conference on machine learning. PmLR, 2021: 8748-8763.

[3] Liu Y, Ott M, Goyal N, et al. Roberta: A robustly optimized bert pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.

[4] Zeiler M D, Fergus R. Visualizing and understanding convolutional networks[C]//Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13. Springer International Publishing, 2014: 818-833.
[5] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.
[6] Bai L, Zhao Y, Huang X. A CNN accelerator on FPGA using depthwise separable convolution[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2018, 65(10): 1415-1419.
[7] Lian X, Liu Z, Song Z, et al. High-performance FPGA-based CNN accelerator with block-floating-point arithmetic[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019, 27(8): 1874-1885.
[8] Mittal S. A survey of FPGA-based accelerators for convolutional neural networks[J]. Neural computing and applications, 2020, 32(4): 1109-1139.
[9] Venieris S I, Kouris A, Bouganis C S. Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions[J]. ACM Computing Surveys (CSUR), 2018, 51(3): 1-39.
[10] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: Transformers for image recognition at scale[J]. arXiv preprint arXiv:2010.11929, 2020.
[11] Yu J, Park J, Park S, et al. NN-LUT: Neural approximation of non-linear operations for efficient transformer inference[C]//Proceedings of the 59th ACM/IEEE Design Automation Conference. 2022: 577-582.
[12] Li Z, Sun M, Lu A, et al. Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization[C]//2022 32nd International Conference on Field-Programmable Logic and Applications (FPL). IEEE, 2022: 109-116.
[13] Li Z, Gu Q. I-vit: Integer-only quantization for efficient vision transformer inference[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023: 17065-17075.
[14] Jacob B, Kligys S, Chen B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C] //Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 2704-2713.
[15] Lin Y, Zhang T, Sun P, et al. Fq-vit: Fully quantized vision transformer without retraining[J]. arXiv preprint arXiv:2111.13824, 2021, 1.
[16] Liu Z, Li G, Cheng J. Hardware acceleration of fully quantized bert for efficient natural language processing[C]//2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2021: 513-516.
[17] Wang H, Zhang Z, Han S. Spatten: Efficient sparse attention architecture with cascade token and head pruning[C]//2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021: 97-110.