國立清華大學 電機工程學系

實作專題研究成果摘要

An Efficient Hardware-Friendly Design of Vision Transformer With Integer-Only Computation and Optimized Tiling

基於整數運算與最佳化 Tiling 的高效且
硬體友善 Vision Transformer 設計

專題領域： 系統領域

組　別 ： A501

指導教授： 黃稚存 教授

組員姓名： 張朝威

研究期間： 2024年7月1日至 2025年5月1日止，共10個月

Contents

摘要 ii

Abstract iii

1 Introduction 1

2 Methodology 2

2-1 An overview: Vision Transformer(ViT)[8-10] 2

2-1.1 Multi-Head Self-Attention (MSA) 2

2-1.2 Feed-Forward Network (FFN) 3

2-2 An overview: Hardware Implementation of ViT 3

2-3 Hardware Implementation of Exponential Function 4

2-4 Hardware Implementation of SoftMax 5

2-5 Hardware Implementation of MatMul 6

3 Experimental Result 7

3-1 Software simulation 7

3-2 Hardware simulation of submodule design 7

3-3 Hardware simulation of MSA module design 8

Conclusion 9

References 10

摘要

Vision Transformer(ViT)近年來已成為與卷積神經網路(CNN)齊名的影像分類架構。然而，Transformer 相較於 CNN 結構更為複雜且資源需求較高，因此許多研究致力於降低其計算複雜度。I-ViT 為首篇完整以整數方式近似 ViT 中所有非線性函數的研究，不僅整合了先前提出的 LayerNorm 近似方法，更透過以 2 為基底的轉換簡化了指數函數的實現，並提供了 SoftMax 與 GeLU 的一種新的近似方式。[1]

然而，I-ViT 的近似設計仍包含部分不利於硬體實作的操作，例如倒數計算與大量的乘除法運算。本研究在 I-ViT 架構的基礎上，提出一個針對 Vision Transformer (ViT) 中 Multi-Head Self-Attention（MSA）部分的 FPGA 硬體友善設計。延續 I-ViT 的資料格式，本研究採用 8 位元整數的量化方法，並針對指數與除法等複雜運算進行簡化與優化。在指數函數部分，本研究提出三級管線指數模組架構以加速運算。同時，使用七個平行指數模組，實現僅用 66 個時脈週期便能完成 196 筆資料的 SoftMax 運算。在矩陣乘法方面，我們利用 tiling 方式對矩陣分割並設計特殊的時序安排，透過優化的 MatMul 硬體設計，同時處理 Q×K 與 SoftMax(QK)×V 的乘法，有效提升整體運算效率。

關鍵字: Transformer, 硬體加速, 量化, 低精度近似

Abstract

In recent years, Vision Transformer(ViT) has emerged as a leading architecture for image classification, rivaling Convolutional Neural Networks (CNNs). However, compared to CNNs, Transformers are more complex and resource-intensive, prompting many studies focused on reducing their computational complexity. I-ViT is the first work to fully approximate all nonlinear functions in ViT using integer-only operations. It not only integrates previously proposed LayerNorm approximations but also simplifies the implementation of exponential functions using base-2 transformations and provides a new approximation methods for SoftMax and GeLU functions. [1]

Despite these advancements, I-ViT still contains certain operations that are not hardware-friendly, such as reciprocal computations and extensive use of multiplication and division. Building on the I-ViT framework, this study proposes a hardware-efficient FPGA design targeting the Multi-Head Self-Attention mechanism in ViT. We adopt the same 8-bit integer quantization format used in I-ViT and further simplify and optimize the operations such as exponentiation and division.

For the exponential function, we propose a 3-stages pipelined exponential modules architecture to accelerate computation and employ seven parallel exponential modules, enabling SoftMax computation on 196 inputs in just 66 clock cycles. For matrix multiplication, we apply tiling techniques to partition matrices and proposed a dedicated timing schedule that allow the system to process Q×K and SoftMax(QK)×V operations concurrently, enhancing overall computational efficiency.

Keywords: Transformer, Hardware Accelerator, Quantization, low-precision approximation

1 Introduction

In recent years, Vision Transformer(ViT), which applies the Transformer architecture to image classification by treating an image as a sequence of patches rather than using traditional convolutional neural networks, have achieved state-of-art performance on image classification tasks. However, their high computational cost and memory requirements lead to significant challenges for hardware implementation, particularly on resource-constrained platforms such as FPGAs and edge devices.

To address these limitations, several quantization methods have been proposed to optimize ViT inference for efficient hardware execution. Notable approaches include powers-of-two scale quantization and log-int quantization for LayerNorm and Softmax in FQ-ViT[2], as well as the data-free quantization method introduced in PSAQ-ViT.[3, 4] However, most of the quantization methods mentioned above rely on floating-point operations and requires substantial computational power and memory bandwidth. Some works attempt to develop architectures with fixed-point-only arithmetic, making the design more hardware-friendly. For instance, I-BERT employs polynomial approximation for non-linear functions[5, 6], and Fully8-bit employs integer-only inference for L1 LayerNorm.[7]

One integer-only quantization architecture, proposed by I-ViT, replaces most complex non-linear arithmetic with linear approximations and simplifies computations using shift operations, significantly reducing computational complexity.[1] However, despite these optimizations, I-ViT still retains reciprocal and division operations in its partial non-linear approximation, which introduces inefficiencies in hardware implementation.

Hence, this work aims to further enhance integer-only ViT inference by proposing a more hardware-efficient architecture with the following key contributions:

- **Optimized Multi-Head Self-Attention (MSA) in I-ViT:** Eliminating the reciprocal-based operations present in I-ViT and reducing the number of division operations to improve computational efficiency.
- **Optimized Tiling-Based Method:** Propose an efficient tiling strategy that effectively partitions the computation workload, optimizing memory access patterns and improving hardware execution efficiency while maintaining the same level of model performance.

By refining these aspects, this study aims to further bridge the gap between high-performance deep learning models and efficient hardware-friendly implementations.

2 Methodology

2-1 An overview: Vision Transformer(ViT)[8-10]

Vision Transformer is one popular deep learning model that applies the transformer architecture to image classification tasks, leveraging self-attention mechanisms to capture global dependencies across image patches. Unlike CNNs, which rely on local receptive fields, ViT treats an image as a sequence of patches and processes them using Multi-Head Self-Attention (MSA) and a Feed-Forward Network (FFN) within the transformer encoder as shown in Fig1.

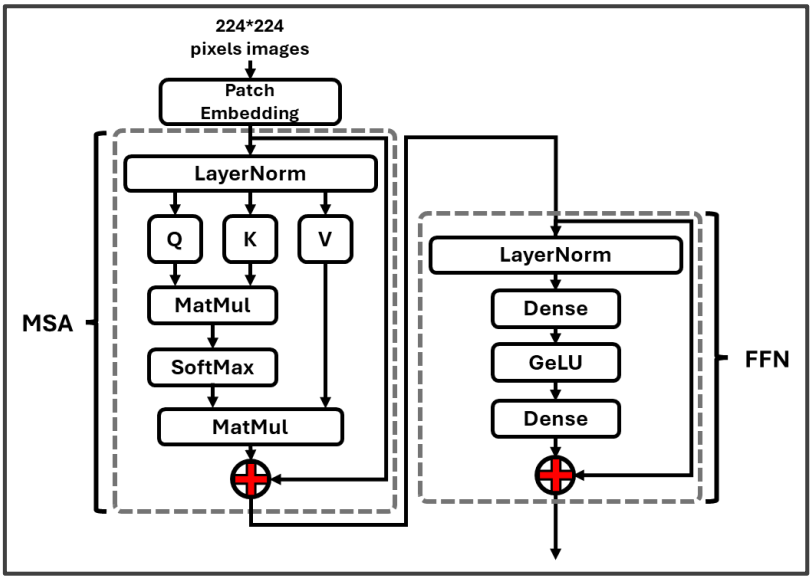

Fig1. The fundamental Architecture of Vision Transformer. The input image is first divided into non-overlapping patches ,and embedded linearly with positional encodings. These tokenized patches are then fed into a stack of Transformer encoder layers, consisting of MSA and FFN. The Pre-Normalization method and the residual connection are applied in this model to mitigate the model degradation and ensuring efficient convergence.

2-1.1 Multi-Head Self-Attention (MSA)

Consider a ViT model with h heads, and given an input sequence of patch embeddings $X \in \mathbb{R}^{N\times D}$, where N is the number of patches and D is the embedding dimension. The three input matrixs query(Q), key(K), and value(V) obtained by linear transformations are prepared as the input of the self-attention mechanism:

$$\begin{cases} Q = XW_Q \\ K = XW_K \\ V = XW_V \end{cases}$$

with learnable weight matrices $W_Q, W_K, W_V \in \mathbb{R}^{N\times d_k}$ and d_k is the number of patches divided by the number of heads.

The self-attention mechanism starts with the inner products of Q and K matrix. Then, the Softmax operation is applied on the product result. Then, the result of previous operation product with V matrix is treated as the output of the self-attention mechanism. Hence, the self-attention mechanism can be represented as the following formula:

$$\text{Attention}(\text{Q}, \text{K}, \text{V}) = \text{Softmax}\left(QK^T \Big/ \sqrt{d_k}\right)\text{V}$$

The division by $\sqrt{d_k}$ ensures numerical stability by scaling the dot product.

Finally, combine the result of multiple self-attention operations into the outpu of multi-head self-attention through a weight matrix $W_{concat} \in \mathbb{R}^{hd_k \times D}$:

$$MSA(X) = Concat(head_1, \dots, head_h)W_{concat}$$

2-1.2 Feed-Forward Network (FFN)

Following the MSA module, ViT employs a Feed-Forward Network to further process the extracted features. The FFN consists of two fully connected layers with a non-linear GELU activation function in between:

$$\text{FFN}(X) = GELU(XW_1 + bias_1)W_2 + bias_2$$

where X is the output of the MSA after the LayerNorm process with residual connection, $W_1 \in \mathbb{R}^{D \times D_{hidden}}$ and $W_2 \in \mathbb{R}^{D_{hidden} \times D}$ are weight matrices, $bias_1$ and $bias_2$ are bias terms. The FFN enhances the model's expressiveness by applying non-linear transformations to the feature representations.

Both the MSA and FFN components are wrapped within residual connections and layer normalization:

$$X' = LayerNorm(X + MSA(X))$$
$$X'' = LayerNorm(X' + MSA(X'))$$

where layer normalization stabilizes training by normalizing activations across feature dimensions.

2-2 An overview: Hardware Implementation of ViT

This work focuses on the hardware implementation and optimization of the MSA module in ViT. The proposed system hierarchy, shown in Fig. 2, is designed to efficiently process input tokens through a structured pipeline. First, the input tokens are multiplied by their corresponding weights using the PE block, after which the resulting Q, K, and V matrices are stored in registers. The designed MatMul module (Sec. 2-5) then performs the QK^T matrix operation, feeding the output into a non-linear SoftMax or GeLU function(Sec. 2-2 and 2-4) before executing the final matrix multiplication.

To maintain the residual connections in the original ViT algorithm, an adder combines the final MatMul output with the original input tokens. Additionally, for greater efficiency, the proposed system hierarchy is designed to support both the MSA(Fig. 2(a)) and FFN(Fig. 2(b))

layers in ViT, integrating both the SoftMax and GeLU blocks. A more detailed design of each submodule is provided in Sections 2.3 to 2.5.

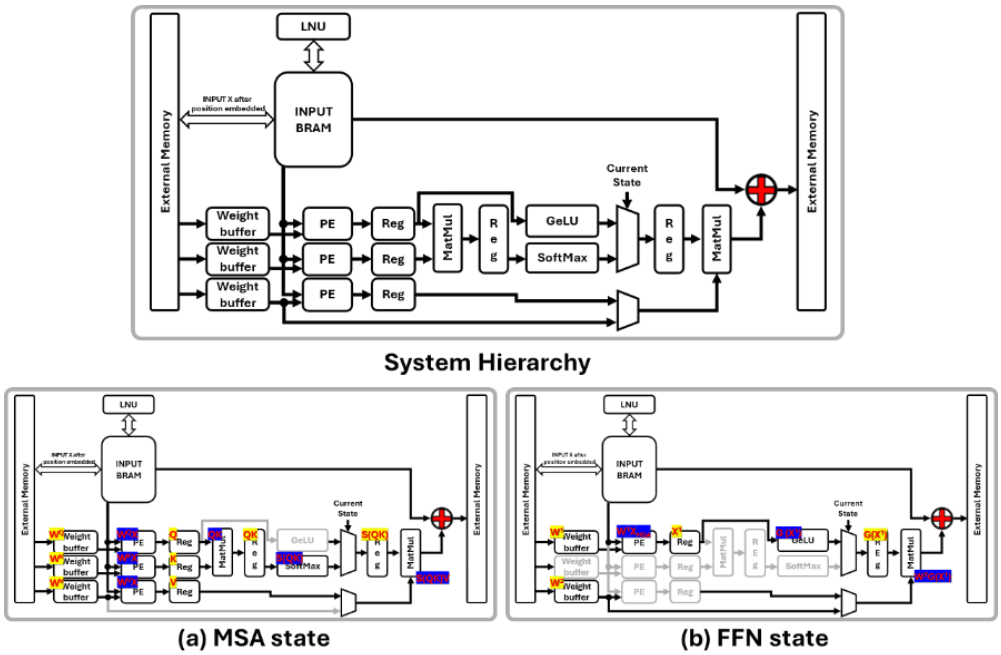

Fig2. System Hierarchy. The proposed hardware architecture for ViT supports both the MSA and FFN modules. The top section presents the overall system hierarchy. Subfigure (a) illustrates the data flow in the MSA stage, where input tokens are processed to generate Q, K, and V, followed by matrix multiplication and non-linear transformations. Subfigure (b) shows the data flow in the FFN stage.

2-3 Hardware Implementation of Exponential Function

Traditional methods for implementing exponential functions in digital circuits typically rely on lookup tables or polynomial approximation methods[5, 6]. While LUT-based approaches enable direct function value lookups, they require substantial hardware resources, making them inefficient for resource-constrained designs. Alternatively, polynomial approximation methods utilize polynomial expansions to approximate exponentiation but necessitate multiple multipliers, leading to high computational complexity and increased hardware utilization.

To address these challenges, various low-precision approximation techniques have been developed based on the logarithmic-exponential transformation.[1, 11] Considering the hardware resource limitations on FPGA, an efficient hardware-aware implementation is required. The low-precision approximation method is applied in this study leveraging exponential-logarithmic base conversion to achieve a hardware-friendly and resource-efficient design.

Consider the input to the exponential function is quantized into INT8 format with scaling factor SF. The first step is to convert the base of the target result from e to 2. The multiplication in the exponent term can be simply performed by shifting operation since $\log_2 e$ can be

approximated by binary as $(1.0111)_2$.[1] The calculation can be represented as

$$e^{SF_\Delta I_\Delta} = 2^{SF_\Delta I_\Delta log_2 e} \approx 2^{SF_\Delta (I_\Delta + I_\Delta \gg 1 - I_\Delta \gg 4)} = 2^{SF_\Delta I_p}$$

Since the result of the multiplication in the exponent term may not be an integer, it cannot be directly used for shifting. To overcome this, the exponent term $S_\Delta I_p$ is decomposed into an integer part q and a decimal part r as follows, simplifying the computation as follows :

$$2^{S_\Delta I_p} = 2^{-(q+r)} = 2^{-r} \gg q$$

where both q and r are positive.

To further reduce the complexity of calculation, the term of 2^{-r} can be approximated as a linear function $-r/2 + 1$ if$-r \in (-1,0]$. Then, according to the relation derived above, the exponential function can be approximated as

$$e^{S_\Delta I_\Delta} \approx 2^{S_\Delta I_p} = 2^{-r} \gg q \approx (-r \gg 2 + 1) \gg q$$

2-4 Hardware Implementation of SoftMax

In ViT, the row-wise SoftMax is applied after the matrix multiplication of the Q and K matrices. Since the input dimension of SoftMax in the DeiT-Tiny model—used as the evaluation benchmark in this study — is 196 × 196, the SoftMax function processes 196 elements per row. However, directly loading all 196 8-bit numbers at once is not feasible for hardware implementation due to resource constraints. To address this, the 196 tokens are divided into 28 groups, with each group containing 7 elements. Recalling the mathematical representation of the SoftMax function, it can be expressed as follows:

$$\text{SoftMax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{S_{x_i} I_{x_i}}}{\sum_j e^{S_{x_j} I_{x_j}}}$$

To smooth the data distribution and prevent overflow, divide the numerator and denominator by I_{max}, which was recorded in the MatMul process stage. The modified SoftMax function can be represented as

$$\text{SoftMax}(x_i) = \frac{e^{S_{x_i}(I_{x_i} - I_{max})}}{\sum_j e^{S_{x_j}(I_{x_j} - I_{max})}}$$

By applying seven exponential modules introduced in Section 2-3, the exponentiation can be efficiently implemented in parallel using integer-only arithmetic. Notably, with this method, the SoftMax function can be realized using only adders, a small number of dividers, and shift operations, significantly reducing computational complexity.

The design is divided into two states: the accumulation state(ACC) and the division state (DIV). In the ACC state, the inputs are first processed by the exponential modules after subtracting I_{max}, and the resulting exponentiated values are accumulated using an adder tree. Due to the limitations of memory bandwidth and hardware reuse, the exponential of the DIV

state as the numerator is not saved in the ACC state, but directly calculated again in the DIV state through the exponential module. Furthermore, to reduce extra power consumption, the division unit is disabled via the div_en control signal during this phase.

2-5 Hardware Implementation of MatMul

Consider the multiplication of two matrices, M_1 and M_2 is the output matrix O. Due to the linearity of matrix multiplication, the quantized input and scaling factors can be computed separately:

$$\mathrm{O} = SF_{\mathrm{O}} I_{\mathrm{O}} = \mathrm{M}_1 \times \mathrm{M}_2 = SF_{\mathrm{M}_1} I_{\mathrm{M}_1} \times SF_{\mathrm{M}_2} I_{\mathrm{M}_2} = \left(SF_{\mathrm{M}_1} \times SF_{\mathrm{M}_2}\right)\left(I_{\mathrm{M}_1} \times I_{\mathrm{M}_2}\right)$$

where I_{M_1} and I_{M_2} are the INT8 matrix and SF_{M_1} and SF_{M_2} remain the floating point datatype. The result of matrix multiplication O is INT32 formation and will be re-quantilization back to INT8 formation.

$$\mathrm{O} = SF_{\mathrm{O}} I_{\mathrm{O(INT8)}} = SF'_{\mathrm{O}} I'_{\mathrm{O(INT32)}} = \left(SF_{\mathrm{M}_1} \times SF_{\mathrm{M}_2}\right)\left(I_{\mathrm{M}_1} \times I_{\mathrm{M}_2}\right)$$

By applying the dyadic arithmetic pipeline, which approximates floating-point scaling factor operations by integer bit shifting, scaling factor multiplication can be executed using integer-only arithmetic:

$$I_{\mathrm{O(INT8)}} = \left\lfloor \frac{SF_{\mathrm{M}_1} \times SF_{\mathrm{M}_2}}{SF_{\mathrm{O}}} \left(I_{\mathrm{M}_1} \times I_{\mathrm{M}_2}\right) \right\rfloor = \left\lfloor \frac{b}{2^c} \left(I_{\mathrm{M}_1} \times I_{\mathrm{M}_2}\right) \right\rfloor$$

where b and c are both positive integer values. Then, the matrix multiplication can be calculated by integer-only arithmetic as following:[1]

$$\mathrm{O} = \left(b \cdot \left(I_{\mathrm{M}_1} \times I_{\mathrm{M}_2}\right)\right) \gg c$$

For hardware implementation, partial sum accumulation and the tiling method are commonly used to efficiently perform matrix multiplication. Inspired by Column-Wise MatMul Algorithm proposed in Co-design of Attention Mechanism[12], this study adopts the row-wise MatMul Algorithm, as it better aligns with the preceding and succeeding stages that involve row-based operations.

Firstly, consider the matrix dimensions involved in the MatMul of the deit_tiny model. For instance, the multiplication could be between matrices of size 196 × 196 or 196 × 64 and 64 × 196. The designed MatMul module processes four rows simultaneously. In each cycle, it loads four elements from the same column of matrix M_1 and seven elements from the same row of matrix M_2 to perform multiply-accumulate operations.

Once all elements in a row of M_1 are multiplied with the corresponding elements in a column of M_2, a single output element in the result matrix O is computed. This accumulated result, held at the output of the adder tree, is then output along with a valid signal indicating completion.

In the DeiT-Tiny model, the input dimensions for MatMul are 196 × 64 and 196 × 196. Since a subsequent step in the computation involves row-wise operations, a row-wise MatMul module is proposed, as shown in Fig.3 .

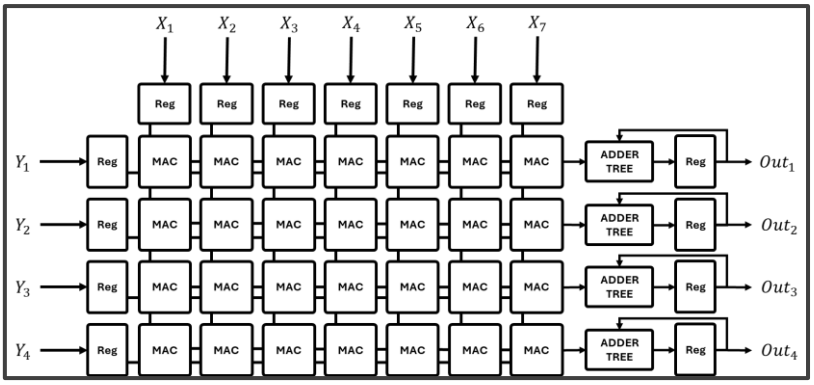

Fig.3 Proposed MatMul module. Four elements from a column of M_1 and seven elements from a row of M_2 are loaded simultaneously. Partial results are processed through MAC units and accumulated via an adder tree. Four output rows are computed in parallel to enhance throughput.

3 Experimental Result

3-1 Software simulation

In the software simulation, we used the Fashion MNIST dataset to train the model with a batch size of 128, a learning rate of 5e-7, momentum of 0.9, and weight decay of 0.0001 for 30 epochs. Without any integer approximation, the model achieved an accuracy of 76.3%. Using the I-ViT architecture[1], the accuracy improved to 77.91%. Our hardware-friendly modification achieved a slightly lower accuracy of 77.0%, representing a drop of about 0.9% compared to I-ViT. However, our model eliminates reciprocal operations in the exponential module and significantly reduces the number of required multiplication and division units.

3-2 Hardware simulation of submodule design

This study aimed to verify the design on FPGA using Xilinx Vivado 2017.4 for simulation and synthesis. However, due to time constraints, only simulation was performed and no verification was performed on FPGA.

In the exponential function, we adopted a 3-stages pipeline architecture. Since the input is an 8-bit integer containing only negative values, we first generated the golden patterns for all 128 input combinations using Python. We then verified the hardware outputs against these reference values.

Similarly, we generate test data through Python to verify the SoftMax function. By utilizing seven exponential modules in a 3-stages pipeline architecture, the system can complete the SoftMax operation for 196 values in just 66 cycles. The designed SoftMax module operates in two main states: ACC and DIV. The ACC state consumes 28 cycles, during which 7 values are loaded at a time, their exponentials are computed, and the results are accumulated to form the SoftMax denominator. In the following 28 cycles, the DIV state divides each value

by the accumulated denominator from the previous stage. Additionally, considering the 5-cycle latency introduced by the DSP block multipliers, the total latency for completing the SoftMax operation remains 66 cycles.

In the MatMul function, Regarding the acceleration of matrix multiplication, using the tiling method, we calculate the elements of four different rows in the output matrix simultaneously. Since 7 elements are loaded at the same time for calculation, the MatMul module requires 1793 cycles to complete the operation for the one rows of values output by the Q and K matrices, and 5489 cycles to complete the operation for the one rows of values output by the SoftMax(QK) and the V matrix.

3-3 Hardware simulation of MSA module design

As mentioned in Section 3-2, the time required to complete a row of MatMul operations differs by approximately a factor of three. To address this imbalance, we propose a timing schedule, as illustrated in Fig.4. In this schedule, three MatMul modules are utilized, one is dedicated to the multiplication of the Q and K matrices, while the other two operate concurrently to perform the multiplication of SoftMax(QK) and the V matrix. This parallel execution effectively optimizes the overall timing arrangement and significantly improves computational efficiency.

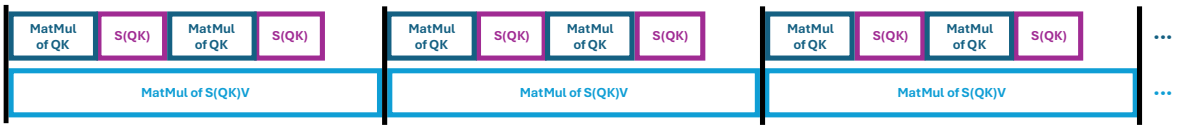

Fig.4 Timing schedule of MSA operations. In this schedule, three MatMul modules are utilized : one MatMul module computes Q × K, while the other two concurrently handle SoftMax(QK) × V.

The hardware resource utilization is shown in Table 1 for each major submodule in the proposed Multi-Head Self-Attention design. Due to the complex approximate of exponential and normalization operations , the SoftMax module accounts for the majority of logic and flip-flop consumption, occupying over 90% of hardware resource in the MSA design.

Table 1 Hardware Utilization

	SoftMax	QK Matmul	QKV Matmul	MSA-Module
LUT	25212	890	822	28623
LUTRAM	21	-	-	21
FF	26708	220	188	27495
DSP	14	44	44	190

Conclusion

This work presents a hardware-friendly MSA accelerator based on a modified version of the I-ViT approximation.[1] Using the Fashion MNIST dataset for software validation, the quantized and optimized model achieved an accuracy of **77.0%**, with only a slight reduction of **0.9%** compared to the original I-ViT architecture (**77.91%**) on the DeiT-Tiny baseline. Notably, the design eliminates reciprocal operations and greatly simplifies the multiplication and division processes, enhancing hardware efficiency.

In the proposed hardware architecture illustrated in Fig.2, the MSA and FFN are shared between both the MSA and FFN same computing architecture, reducing hardware redundancy compared to traditional architectures that allocate separate resources for each stage.

Despite these achievements, the design still faces certain limitations when compared to more advanced accelerators. The SoftMax module alone consumes over **90%** of the total logic and flip-flop resources, suggesting that further optimizations could substantially reduce area and power without significantly compromising accuracy.

Additionally, the current implementation focuses only on the MSA component. However, based on the proposed shared architecture for MatMul and nonlinear modules, we can reasonably estimate the performance for the full ViT pipeline. At 150 MHz, our MSA accelerator completes the computation for one head in 0.632 ms. Given that the DeiT-Tiny model uses 3 heads, the full MSA stage would require approximately 1.896 ms. Although the FFN stage hardware is not yet fully implemented, we estimate the FFN would require an additional 0.621 ms based on the MSA and FFN shared architecture. Thus, the total processing time for one image would be approximately 2.517 ms.

In comparison, running the original FP32 DeiT-Tiny model on an RTX 2080Ti GPU requires around 5.99 ms per image, while the I-ViT integer-approximated model achieves inference in 1.61 ms.[1]Although our hardware accelerator improves inference speed by roughly 2.34× over the original model, which seems less than the 3.72× achieved by I-ViT, this discrepancy is expected. In software, all attention heads can be computed in parallel, while in our hardware accelerator, the heads are computed sequentially due to resource sharing and pipeline scheduling. Therefore, the direct comparison of runtime is not entirely one-to-one, and our design still offers a promising balance between hardware cost, modular reuse, and performance. Future work will focus on fully integrating the ViT accelerator, optimizing across all stages, and improving throughput and resource efficiency.

References

[1] Z. Li and Q. Gu, "I-vit: Integer-only quantization for efficient vision transformer inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17065-17075.

[2] Y. Lin, T. Zhang, P. Sun, Z. Li, and S. Zhou, "Fq-vit: Post-training quantization for fully quantized vision transformer," *arXiv preprint arXiv:2111.13824,* 2021.

[3] Z. Li, M. Chen, J. Xiao, and Q. Gu, "Psaq-vit v2: Toward accurate and general data-free quantization for vision transformers," *IEEE Transactions on Neural Networks and Learning Systems,* 2023.

[4] Z. Li, L. Ma, M. Chen, J. Xiao, and Q. Gu, "Patch similarity aware data-free quantization for vision transformers," in *European conference on computer vision*, 2022: Springer, pp. 154-170.

[5] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in *International conference on machine learning*, 2021: PMLR, pp. 5506-5518.

[6] A. Marchisio, D. Dura, M. Capra, M. Martina, G. Masera, and M. Shafique, "SwiftTron: An efficient hardware accelerator for quantized transformers," in *2023 International Joint Conference on Neural Networks (IJCNN)*, 2023: IEEE, pp. 1-9.

[7] Y. Lin, Y. Li, T. Liu, T. Xiao, T. Liu, and J. Zhu, "Towards fully 8-bit integer inference for the transformer model," *arXiv preprint arXiv:2009.08034,* 2020.

[8] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929,* 2020.

[9] K. Han *et al.*, "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence,* vol. 45, no. 1, pp. 87-110, 2022.

[10] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.

[11] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," in *2018 IEEE asia pacific conference on circuits and systems (APCCAS)*, 2018: IEEE, pp. 223-226.

[12] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on FPGA devices," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 20, no. 5s, pp. 1-24, 2021.

Reflection and Thoughts

First and foremost, I would like to express my sincere gratitude to Prof. Chih-Tsun Huang for his invaluable guidance throughout the past year. His mentorship has been instrumental in helping me cultivate a more rigorous and structured approach to both literature review and hardware system design.

The original objective of this project was to develop a complete hardware accelerator for the ViT model and validate its performance on an FPGA platform. Due to time constraints, only the RTL verification of the MSA module was completed, with the FFN module left for future implementation. Nevertheless, we designed a unified and reusable architecture capable of supporting both MSA and FFN modules, laying a robust foundation for further development.

This study centers on a hardware-friendly implementation of ViT, leveraging integer-only computation and an optimized tiling strategy to enhance both efficiency and practicality. During this project, I investigated the computational bottlenecks of ViT, especially the high memory bandwidth and arithmetic intensity associated with the MSA mechanism.

While reviewing existing quantization strategies, I found I-ViT to be an effective solution for reducing design complexity. However, its reliance on reciprocal operations and other hardware-unfriendly functions posed limitations. Building on I-ViT's integer quantization scheme, this work simplifies the circuit by eliminating reciprocal operations and introducing a refined tiling method to reduce memory access while maintaining parallelism and throughput.

Through this research, I gained valuable experience in digital hardware design, particularly in balancing trade-offs between accuracy, resource usage, and computational speed. I also deepened my understanding of how to approximate nonlinear functions using pipelined architectures and shift-based operations. Although this project successfully optimized the MSA module for hardware implementation, a key limitation remains in the absence of a completed FFN module. Future work will focus on extending these hardware optimizations to the FFN, ultimately aiming to realize a fully hardware-efficient ViT accelerator.