An Integer-Only and Group-Vector Systolic Accelerator for Efficiently Mapping Vision Transformer on Edge

Mingqiang Huang, Junyi Luo, Chenchen Ding, *Member, IEEE*, Zikun Wei, Sixiao Huang, and Hao Yu, *Senior Member, IEEE*

***Abstract*— Transformer-like network has shown remarkable high performance in both natural language processing and computer vision. However, the huge computational demands in non-linear floating-point arithmetic and the irregular memory access requirement in self-attention mechanism make it still a challenge to deploy Transformer on edge. To address the above issues, we propose integer-only quantization scheme for the simplification of non-linear operations (such as LayerNorm, Softmax and Gelu), meanwhile algorithm-hardware co-design strategy is applied to guarantee both the high accuracy and high efficiency. Besides, we construct general-purpose group vector systolic array to efficiently accelerate the matrix multiplication operations including both regular matrix-multiplication/convolution and the irregular multi-head self-attention mechanism. Unified data-package strategy and flexible on-/off-chip data storage management strategy are also proposed to further improve the performance. The design has been deployed on Xilinx ZCU102 FPGA platform, achieving an overall inference latency of 4.077ms and 11.15ms per image for ViT-tiny and ViT-s, respectively. The average throughput can reach as high as 762.7 GOPs, which shows significant improvement over the previous state-of-the-art FPGA Transformer accelerator.**

***Index Terms*— Integer-only transformer, systolic accelerator.**

I. Introduction

TRANSFORMER network has demonstrated remarkable success in natural language processing [1], [2], [3], leading to renewed interest in utilizing this architecture for computer vision applications [4], [5], [6], [7]. By treating an image as a sequence of patches and processing it with a standard Transformer encoder, the resulting Vision Transformer (ViT) has achieved or surpassed state-of-the-art results on numerous image classification datasets [4]. In comparison to traditional convolutional neural networks (CNN) [8], [9], which have proven effective in hierarchical feature extraction but may struggle with modeling long-range dependencies, the Transformer overcomes this limitation by using the self-attention mechanism and allows for parallelization across the input sequence. As a result, the ViT architecture offers improved efficiency, using approximately 2-4 times less compute power to achieve equivalent performance than CNNs [9], [10]. FPGA implementation of ViT has the potential to significantly improve the efficiency, but there are still some challenges to be addressed.

One of the primary challenges in accelerating ViT is the design of the nonlinear and floating-point functions. Unlike traditional CNN networks, which predominantly utilize linear operations and can be efficiently quantized through low-bit quantization techniques, the Transformer networks heavily rely on complex nonlinear operations such as LayerNorm, Softmax, and Gelu activation [4], [10]. These operations have been incorporated into the ViT architecture to improve performance. For example, LayerNorm is applied to each layer of the model after the self-attention mechanism or the Multi-Layer Perceptron (MLP) sub-layers, and serves to normalize the output of each layer to have zero mean and unit variance. Gelu activation, on the other hand, is based on the Gaussian error function, providing a smooth, bell-shaped curve. While these nonlinear functions contribute to increased robustness and performance, they also result in computationally intensive operations due to the floating-point arithmetic. One potential solution is integer quantization. However, previous research mainly focused on software-based simulation quantization, or "fake quantization", in which only the inputs and outputs are integer values, while the actual compute-intensive operations are still performed using floating-point values. This approach does little to reduce computational costs and therefore has limited impact on model acceleration. A recent study [11] proposed integer-only quantization for ViT, fully quantizing the model and allowing for the entire inference to be performed with integer arithmetic and bit-shifting. However, the design is based on GPU platform and the quantization scheme still requires lots of division operations, which may not be suitable for resource-limited edge devices.

The second challenge that arises in Transformer-like network is related to the multi-head self-attention mechanism. In Transformer, attention is a mechanism to find the words of importance for a given query word in a sentence, which

Manuscript received 8 May 2023; revised 14 August 2023; accepted 26 August 2023. Date of publication 19 October 2023; date of current version 18 December 2023. This work was supported in part by the STI 2030-Major Project 2022ZD0210600, in part by the National Natural Science Foundation of China (NSFC) through the Key Program under Grant 62034007, and in part by the Shenzhen Science and Technology Program under Grant KQTD20200820113051096 and Grant JCYJ20200109115210307. This article was recommended by Associate Editor J. Di. *(Corresponding author: Hao Yu.)*

Mingqiang Huang is with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China.

Junyi Luo, Chenchen Ding, Zikun Wei, Sixiao Huang, and Hao Yu are with the School of Microelectronics, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: yuh3@sustech.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2023.3312775.

Digital Object Identifier 10.1109/TCSI.2023.3312775

is done by passing the query(Q), key(K) and value (V) using $A= Q^*K^T$ and Out=f(A)*V. On one hand, different from the normal matrix multiplication where the weight parameters can be pretreated, here both K and V are on-line generated. On the other hand, to adapt the parallel computation in PE array, the activation-types of K and V have to be reshaped as weight-type, leading to irregular and inconsecutive memory access requirement. Besides, the multi-head QKV data are all high dimensional tensors, which will undoubtedly further increase the complexity in control of dataflow.

In addition to the nonlinear functions and self-attention mechanism, the Transformer network also includes many other operations, such as convolution, embedding concat, multi-head-based matrix reshape, shortcut, and full-connection layer. To efficiently accelerate the entire Transformer network on a FPGA, it is crucial to develop a unified data package scheme and a parallel data processing strategy for all operations. Partitioning the feature data into smaller segments and processing each segment in parallel can achieve parallel processing in specific modules without data dependency [12], [13], [14], [15], such as Relu and Gelu activations. However, it is not applicable to operations with data dependency, such as Softmax, LayerNorm, and multi-head self-attention (MSA). Moreover, rearranging the high-dimensional tensor data (5D-data: batch, heads, channel, height, width) in hardware design remains an open question. Previous works [16] conducted a thorough analysis and proposed a potential optimization method with two reused processing engines for the internal stream. But this approach only focuses on the shortcut dataflow, leaving several other operations to be optimized.

The last challenge associated with Transformer is the bottleneck in memory access [17]. Memory access is a critical concern for all of the neural networks. Transformer employs a multi-head self-attention mechanism and represents features in high-dimensional space. In contrast to CNNs, where a single convolution kernel filter can slide through the entire image with high data reuse times (ranging from 100 to 10000) [8], [9], the reuse times in Transformer-like networks are relatively small (about several hundreds), resulting in a significant memory access bottleneck. Addressing this challenge requires effective management of memory and efficient data movement across different memory hierarchies.

In this work, we develop an FPGA acceleration framework for integer-only (INT8) quantized ViT network, which shows much higher efficiency than previous accelerator, and retains higher accuracy. The contributions of our work are:

1) We develop INT8 quantization scheme and full hardware design for ViT, which achieves the entire ViT network performing without any floating-point operations.
2) We propose efficient FPGA computing engine with unified data package scheme and group-vector systolic array for data processing, where both regular matrix multiplication and the irregular multi-head self-attention operations can be well performed.
3) We design dynamic and flexible on-chip and off-chip data storage management strategy, and combine different operations together to decrease the memory-access times, thus increasing the efficiency.
4) The proposed method is general-purpose and can be extended to other kinds of Transformer-like accelerators. The whole design has been successfully deployed on Xilinx ZCU102 FPGA platform. The average throughput can reach as high as 762.7 GOPs, which shows significant improvement comparing with the previous state-of-the-art result.

II. Background and Related Work

In this section, we will introduce the architecture of Vision Transformer, the quantization scheme and the hardware design of FPGA Transformer accelerator.

A. Vision Transformer

ViT represents a pioneering attempt to apply Transformer-based models to vision applications, achieving higher accuracy than CNNs in image classification [4]. Its success can be attributed to three main factors. Firstly, Transformers use an attention mechanism to selectively focus on different parts of the input, which can be particularly useful for tasks that require modeling long-range dependencies, such as object detection and image captioning. In contrast, CNNs employ a fixed-size filter across the input and lack an explicit mechanism for capturing long-range dependencies. Secondly, Transformers are more flexible than CNNs in handling inputs of arbitrary sizes and shapes. Thirdly, Transformers are more effective than CNNs for transfer learning across different tasks and domains because they learn more generic and task-agnostic representations. The attention mechanism in Transformers enables them to capture fine-grained relationships between different parts of the input, which can be useful for generalization to new domains. Following the success of ViT, other Transformer-like networks, such as DeiT [10] and T2T-ViT [18], were proposed to reduce dependency on massive pre-training and achieve better accuracy than ResNets [9] of comparable size on ImageNet.

B. Quantization of Vision Transformer

ViT is a highly capable model for image classification and object detection, but its computational and memory requirements are significantly greater than those traditional CNN neural networks. To address this challenge, quantization has been proposed as a solution [11], [19]. Quantization involves the process of representing numbers using a fixed number of bits. In the context of deep learning models such as ViT, quantization entails reducing the precision of the model's weights and activations from 32-bit floating-point numbers to lower-precision numbers, such as 16-bit or 8-bit integers. This technique reduces the memory required to store the model and speeds up computation rates, as operations on fixed-point numbers are more efficient than on floating-point numbers. Additionally, power-of-two (PoT) quantization is another method for quantizing the weights and activations of deep neural networks [20]. PoT quantization utilizes weights and activations represented as powers of two, typically utilizing 2-4 bits per value, resulting in a smaller memory footprint and faster computation. However, the accuracy loss

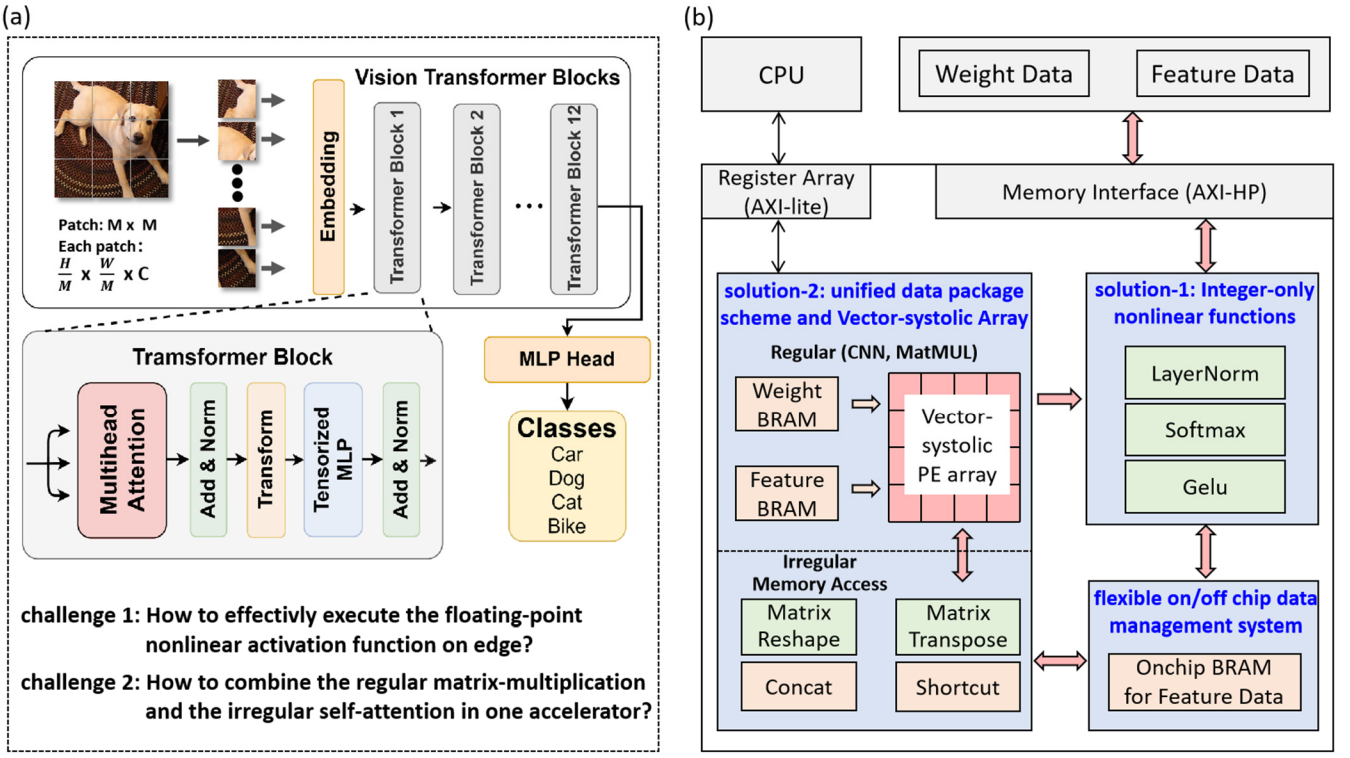

Fig. 1. (a) Architecture of the Vision Transformer network and challenges in hardware design for Transformer-like accelerator. (b) Proposed vision Transformer acceleration framework on FPGA.

incurred through PoT quantization is greater than that of INT8 quantization. For example, the baseline of FP32 DeiT-s shows a high TOP-1 accuracy of 79.85%. In INT8 quantization [11], the accuracy slightly increases up to 80.12%. While in PoT quantization [20], the accuracy drops to be only 77.97%.

C. FPGA Accelerator of Vision Transformer

The implementation of Transformer accelerator on FPGA is significant and necessary because it can greatly accelerate the computation on specialized hardware and reduce power consumption, making it more feasible for real-world applications. Furthermore, FPGA implementation offers flexibility in design, allowing for customization and optimization of the model for specific applications. Currently, there have been some advances in FPGA implementation of ViT. For example, [20] proposes an FPGA-aware automatic ViT acceleration framework based on the fixed point and PoT mixed scheme quantization, in which the accelerator achieves around 56.8 FPS with 0.71% accuracy drop on ImageNet dataset for DeiT-base. However, the PoT quantization sacrifices the accuracy and it can hardly be transferred to other models. Reference [16] proposes the ViA, a novel vision Transformer accelerator architecture based on FPGA, and obtains nearly 309.6 GOP/s computing performance in the peek. They use the half-layer mapping to reduce the impact of path dependence caused by the shortcut mechanism, however, leaving many of other operations to be optimized.

III. Hardware Design of the Nonlinear Operations

In this section, we discuss the integer-only quantization of nonlinear operations including LayerNorm, Softmax, and Gelu. The corresponding hardware design are also presented.

A. INT8 Quantization

In Transformer-like networks, the quantization scheme faces challenges when dealing with non-linear functions such as LayerNorm and Softmax. Although integer polynomial approximation or look-up-table can be good methods for these functions, they are still not efficient for hardware. Recent study [11] proposed an bit-shifting and integer-based approximation arithmetic for the non-linear functions. However, it still involves larger numbers of division operation and may not be suitable for resource-limited edge devices. This work develops integer-only approximate functions to further decrease the hardware consumption. We have re-trained the ViT-tiny network with different approximated approaches for the non-linear functions (Fig.2). Compared with the baseline of [10], the recognition accuracy shows zero loss using our method. Though our accuracy is lower than that using the method in [10], our design is more efficient: the GPU runtime of the key non-linear function Softmax has been decreased by 17.7%, and the total runtime is 4.1%-off, indicating the good efficiency.

B. Hardware Design of the Integer-Only LayerNorm

Layer normalization (LayerNorm) is a technique that has been widely used in Transformers to improve the training and performance of the models [6]. In ViT, LayerNorm is applied to each layer of the model after the self-attention mechanism and the MLP (Multi-Layer Perceptron) sub-layers. The purpose of LayerNorm is to normalize the output of each layer to have zero mean and unit variance, which eliminates the change in the distribution of inputs to each layer during training. This can help with faster and more stable training of the model. Besides, normalizing the output of each layer

model: ViT-tiny	baseline [10]	retrained i-ViT [11]	this work
Top-1 Acc.	74.3%	75.372%	74.386%
Top-5 Acc.	-	92.512%	91.982%
GPU runtime of single Softmax (key function)	-	52.6 us	43.3 us
GPU runtime of the whole ViT model	-	5.38 ms	5.16 ms

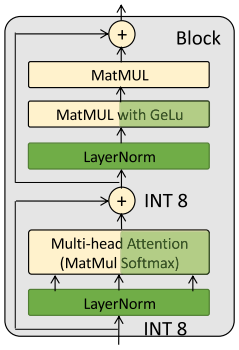

Fig. 2. INT8 Quantization of Vision Transformer and the retrained network performance compared with [11].

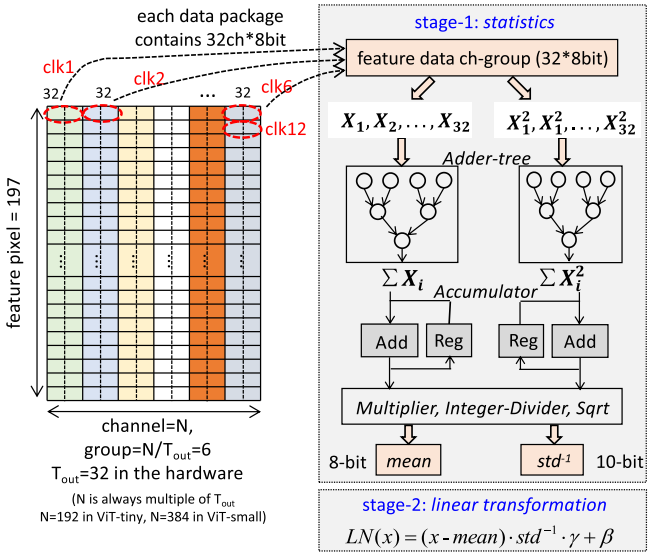

Fig. 3. Integer-only LayerNorm and its hardware design, in which stage-1 is to get the statistics mean and variation, stage-2 is to calculate the LayerNorm.

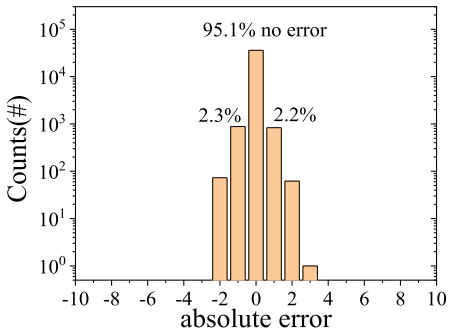

Fig. 4. Error counts for the Integer-only LayerNorm design in monte-carlo analysis.All of the parameters are quantized into 8 bit with fixed point data type. The tested inputs, LN parameters and the corresponding scale value of fixed-point are all random generated. Total case number is 197*192 = 37824.

can reduce the impact of outliers or noisy inputs, which can improve the overall robustness of the model. LayerNorm in ViTs normalizes the input in the hidden feature dimension as

$$LN(x) = \frac{x - Mean(x)}{\sqrt{Var(x) + \varepsilon}} \cdot \gamma + \beta \tag{1}$$

where "Mean(x)" and "Var(x)" represents the statistics of mean value and variance, respectively.

Different from the BatchNorm that holds fixed parameters from training and can be pretreated during inference, LayerNorm needs to dynamically compute the statistics parameters, which is quite hardware unfriendly. According to the definition of variance, it requires fetching the complete data for two times, the first one is for the mean value, the second one is for the difference of (x- mean). Mathematically, it can be simplified to be only once of data access due to

$$Var(x) = E\{[x - E(x)]^2\} = E(x^2) - [E(x)]^2 \tag{2}$$

where "x" is the input data, "E(x)" is the mean value, and "$E(x^2)$" represents the average of squared x. In this way, the variance can be concurrently calculated using the summation of x and summation of x^2. Next, the summation of x and x^2 will be multiplied by integer of 65536/N (N can be 192 or 384 depends on the ViT model) to get the expectation value of "E(x)" and "$E(x^2)$". Finally, the integer Divide and Sqrt (square root) module are designed to calculated the standard variation (std) and its inverse (std^{-1}) for the LN computation.

The overall hardware design of proposed integer-only LayerNorm is shown in Fig. 3. Take the ViT-tiny as an example, the original feature data map is in shape of (pixel, channel) = (197, 192). In this work, we propose unified data-package strategy for all of the Transformer operations with a parallel factor T_{out} =32 along its channel depth dimension. Therefore the actual matrix shape will be (192//T_{out}, 197, T_{out}). According to the definition of LN, the statistics value of "Mean(x)" and "Var(x)" comes from the 192 channels, therefore the hardware will take 192//T_{out} = 6 clock cycles to execute the statics computation for each feature pixel.

The data bit-width in each stage has been carefully designed. According to the definition of LayerNorm, we need to calculate the summation of activations per channel. Take the Vit-tiny as an example, the channel depth is 192, thus the numbert of data to be accumulated is 192. At each clock, the adder tree of x (to get the mean value) will be fed with 32 input data. Since the bit-wdith of x is 8bit, the output of adder tree will be 8+log2(32)=13bit. Afterwards, the output data will be accumulated clock-by-clock until all of the 192 are executed. Therefore the output bit-with of summation-x will be 8+log2(192)=16bit. Similarly, the output bit-with of summation-x2 will be 16+log2(192)=23bit. Remarkabley, the hardware is designed for general-purpose Transformer applicaion, enough margin should be are reserved for the network with more than 192 channel-depth. Here we set the extended datawidth after adder-tree is 8bit, which means the maximum channel-depth is 8192.

In the Division and Sqrt operation, the statistical mean data and variance data will never overflow at 8bit and 16bit, respectively. For the standard variation (std) and its inverse value of std^{-1}, we set them to be 10bit to retain low computation loss. In stage-2, the input features will be performed with the following operations: subtraction, multiplication with 1/std, multiplication with γ, left-shifting to get point alignment with β, add with β, right-shift back and send out. In each step, the bit-width is set to be just enough not to overflow. The result of absolute error counts for the integer-only LayerNorm is shown in Fig.4. Most of the computation are with zero-error due to the fine-tuned bit-width settings.

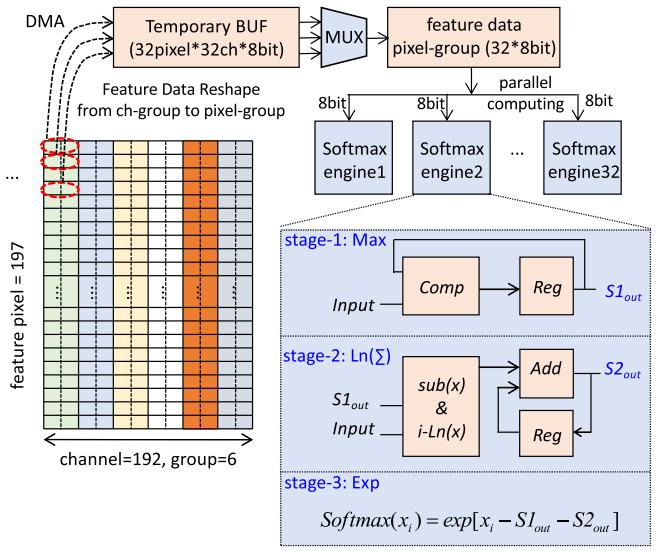

Fig. 5. Integer-only Softmax and its hardware design. The integer-Softmax needs three times of data fetching and all of the functions are linear.

C. Hardware Design of the Integer-Only Softmax

Softmax in Transformer translates the attention scores into probabilities and is calculated by

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum \exp(x_i)} \tag{3}$$

The first challenge is that exponential arithmetic is typically unsupported by integer-only logic. Another drawback of the architecture is the potential overflow problem. To address the above issues, the previous works [11] propose Shiftmax, which can utilize simple hardware logic to achieve accurate and efficient Softmax. However, it still requires lots of division operations. In this work, we further improve the Shiftmax arithmetic and remove the division by using approximated logarithmic function and exponential function.

The key idea is to down-scale the parameters by subtracting the maximum value (X_{max}), and then perform the equivalent transformation using the base changing formula of the exponential function, namely

$$Softmax(x_i) = \exp\{x_i - X_{max} - \ln[\sum_j^N \exp(x_j - X_{max})]\} \tag{4}$$

$$\exp(x) = 2^{[(\log_2^e)\cdot x]} = 2^{u+v} \approx 2^u \cdot (1 + 0.5v) \tag{5}$$

$$\begin{aligned} \ln(x) &= \ln 2 \cdot \log_2 x = \ln 2 \cdot (w + \log_2 k) \\ &\approx \ln 2 \cdot (w + k - 1) \end{aligned} \tag{6}$$

in which the parameters of u, v, w, k can be deduced by using

$$\begin{cases} u = \lceil (\log_2^e) \cdot x \rceil \\ v = (\log_2^e) \cdot x - u \\ k \cdot 2^w = x \\ k \in [1, 2) \end{cases} \tag{7}$$

The above Softmax function and the exponential and logarithmic function can all be approximated by using linear functions such as bit-shift, multiply and add, which significantly improves computational efficiency.

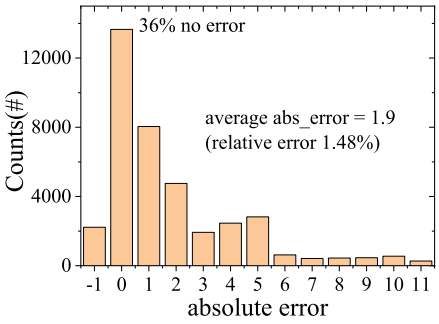

Fig. 6. Absolute error counts for the Integer-only Softmax design. Total case number is 197*192 = 37824.

To complete the whole Integer-Softmax function, it requires three times of data fetching: stage-1 is to find the maximum value of X_{max}, stage-2 is to calculate the summation of exponential function and approximate logarithmic value, and stage-3 is to get the final Softmax result for each input.

Fig.5 shows the detailed hardware design of proposed integer-Softmax. Overall, it contains two modules. First one is the temporary BUF for feature data reshape from ch-group to pixel-group, and the second one is for the parallel computing. As we have discussed before, the packaged feature are in shape of (channel_group, pixel, T_{out}) = (6, 197, 32). In such condition, the 192 channel data (each feature pixel contains 192 channels) are not continuous, which does not meet the demands of Softmax function. Generally speaking, it is not difficult to fetch the 192 channel data pixel-by-pixel, while such operation with discrete address will definitely decrease the system efficiency because it against the Advanced eXtensible Interface (AXI) high performance protocol. To increase the communication efficiency and execute parallel processing, we develop the additional temporary BUF with self-defined DMA (direct memory access) module, by which the 32 features with continues increasing address (along the vertical directions in Fig.5) are fetched clock-by-clock. Since each feature pixel contains 32 channels, the total temporary BUF will be 32*32*8 = 8kb. Next, the 32-pixel feature data will be sent to the Softmax engine array, and each engine receives 8 bit every clock. The integer-only softmax function will be executed in three stages as discussed in equation (3) to equation (7).

The result of absolute error for the integer-only Softmax is shown in Fig.6. The computation error is larger than that of LayerNorm, which mainly comes from the approximation of exponential and logarithmic computation, namely the parameters of u, v, w, k. It can be seen that 36% of the data are with zero-error, and most of the absolute error is less than 10, corresponding to relative error less than 7.8%. The weighted average absolute and relative error is 1.9 and 1.48%, respectively.

D. Hardware Design of the Integer-Only Gelu Activation

Gelu is the non-linear activation function in ViT, which comes from the study of [19]. The hardware design of Gelu can be much simpler than LayerNorm and Softmax because there is not data dependency during the Gelu computation. In consideration of the fact that Gelu is always appeared after

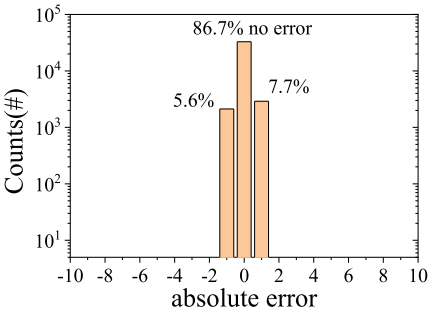

Fig. 7. Absolute error counts for the Integer-only Gelu design.

the matrix computation, we insert the Gelu operation in the corresponding data path. For the Gelu calculation, it can be approximated by using

$$Gelu(x) = x \cdot \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt \approx x \cdot \frac{1}{2}[1 + L(\frac{x}{\sqrt{2}})] \quad (8)$$

where $L(x)$ represents the second-order polynomial approximation by using

$$L(x) = sgn(x)[\alpha(clip(|x|, max = -\beta) + \beta)^2 + 1] \quad (9)$$

in which α = -0.2888, β = -1.769, and "sgn" denotes the sign function. The integer Gelu has an average error of 0.0082 according to the reference [19]. And our experiment test exhibits similar result as shown in Fig.7, indicating slightly better performance than the sigmoid based approximation meanwhile without any floating point arithmetic.

E. Comparison With Previous Work

For Softmax function, both the reference [11] and us use the equivalence transformation method to transfer $\exp(x)$ to the power function of two, namely $2x$. Next, the reference use integer-division to calculate the $\exp(x_i)/\sum(\exp(x_i))$, while we use logarithmic function to convert division in to subtraction, namely A/B=exp[ln(A/B)]=exp[lnA-lnB]. Further, both $\exp(x)$ and $\ln(x)$ have been approximated by several other linear functions. For Gelu function, we use different approximated Gelu functions. Besides, they again use integer-division to calculate the sigmoid function in step-2, meanwhile we use approximation function of $L(x)$ to calculate the final results. For LayerNorm function, they use bit-shifting optimized lightweight integer iterative approach to get the iteration value and calculate the std^{-1}(inverse of standard deviation), while we use and integer-divider to directly get the std^{-1}.

The reason why we use divider in LayerNorm but not in Softmax and Gelu is related with the dataflow in hardware. As we will discuss later, the activation data in accelerator is with T_{out} =32 parallelism in the channel-depth dimension. Therefore, it requires 32 number of dividers in Softmax and Gelu, which will consume too much logical/power resources. While in LayerNorm, the channel-depth dimension will be contracted (accumulated), therefore it needs only one divider, which we believe is acceptable even for the edge device.

IV. Hardware Design for the Matrix Computation

In This Section, We Develop Unified Data Package Scheme and Propose Group-Vector Systolic Array for the Matrix Computation. The Computation Can Be Classified Into Two Types, the First One Is Matrix Multiplication, Convolution and Full-Connection With Regular Memory Access, the Second One Is Multi-Head Self-Attention With Irregular and Inconsecutive Memory Access Requirements.

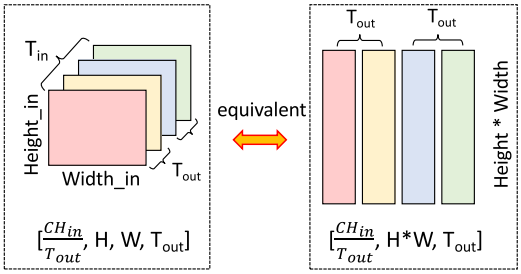

Fig. 8. Unified Data Package Scheme for the feature data. The feature shape in left and right are identical. The left is mainly used in matrix computation to get higher efficiency (see section IV part B), and the right shape is used for the other operations.

A. Unified Data Package Scheme for the Accelerator

There are about 10 kinds of matrix operations in Transformer network, including Convolution (CNN), Concat, Matrix Matrix-multiplication (MatMUL), LayerNorm, Softmax, Multi-head Matrix Reshape, Matrix Transpose, Gelu, and Shortcut. For each operation, the feature data may be in different tensor shapes. To effectively manage the high-dimensional tensor data, we propose the unified data package method for all of the operations (Fig.8). The basic scheme is to group the adjacent channels (T_{out} = 32) together. For example, the packaged feature in LayerNorm or Softmax are in shape of (channel_group, pixel, T_{out}) = (6, 197, 32). For the matrix computation, we can simply split the dimensional of "pixel" into "width" and "height", namely both the input feature and output feature can be organized as (CH//T_{out}, H, W, T_{out}). While for the even higher dimensional tensor data used in the Multi-head Self-Attention module, the shape can be defined as (head_numbers, CH//T_{out}, H, W, T_{out}), which also the same as the other operations. The benefit of such unified data package scheme is that the structure of output is exactly the same as its input in all of the operations, thus we do not need any other software/hardware reshape to re-arrange the features. As a result, the whole network can be effectively accelerated layer-by-layer (or operation-by-operation).

Specifically for the matrix computation, both feature and weight matrix need to be tiled and packaged (Fig. 9). To adapt the parallel computation, each packaged image pixel contains T_{out} numbers of the original image input channels. Since the hardware parallel computation module can get T_{out} results at each clock, the output activation data will also be ordered as (CH/T_{out}, H, W, T_{out}). In such method, the structure of input and output are exactly identical, which is beneficial for the acceleration. The package method for weight is much more complex than feature, and two key points need to be highlighted. On one hand, the size of PE array (or the parallel computation factor) is $T_{in}^{*}T_{out}$, therefore the smallest group of weights should be the same size, namely each weight block contains the required $T_{in}^{*}T_{out}$ weights. On the other hand, the weight data package will be pre-treated and re-ordered

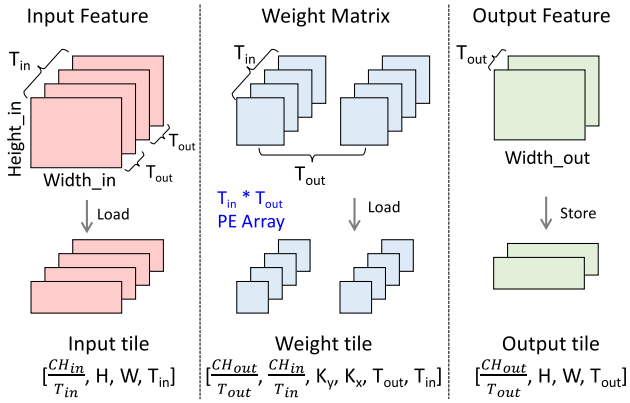

Fig. 9. Data package and ask decomposition in matrix computation.

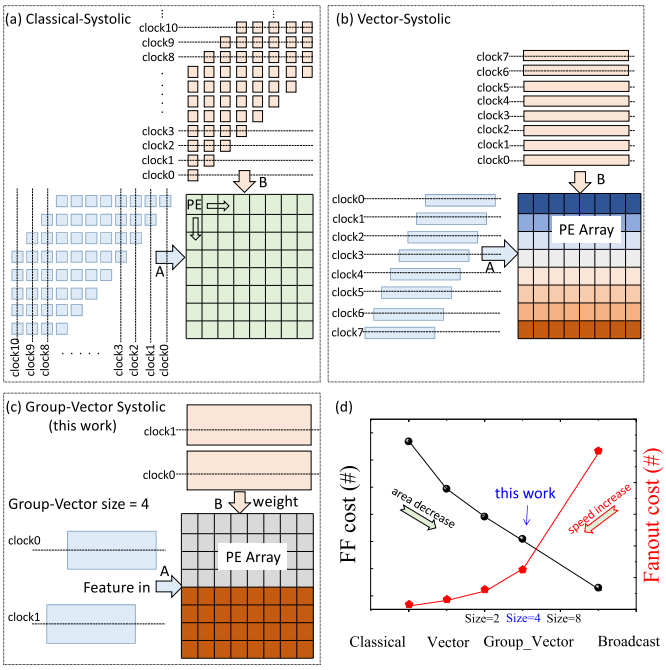

Fig. 10. Detailed dataflow and timeline in (a) Classical systolic array, (b) Vector systolic array, and (c) Group-vector systolic array (this work). (d) FF (Area) and speed cost analysis on different systolic array.

from (CH_{out}, CH_{in}, K_y, K_x) to (CH_{out}/T_{out}, CH_{in}/T_{in}, K_y, K_x, T_{out}, T_{in}). For the matrix multiplication where the kernel size is fixed to be 1, the weight matrix can be simplified to be (CH_{out}/T_{out}, CH_{in}/T_{in}, T_{out}, T_{in}).

B. Group-Vector Systolic Array

In nerual network, >95% of the computation operations are either matrix multiplication or convolution [21], [22], [23], [24]. The computation is usually executed in the process element (PE) array. Google's TPU [21] is the most famous systolic array, which executes the matrix multiplication with a fine-grained pipeline (Fig.10a), thus it can reach great high frequency up to GHz. However, the cost is that it requires huge numbers of registers to store the temporary data. Therefore, the dynamic power and circuit area will be large. To decrease the circuit area and power thus increasing the efficiency, vector systolic array(VSA) is proposed in [23]. Different from TPU or broadcast-style systolic array, the input features and weights in VSA are transferred into PE array in a row-by-row style

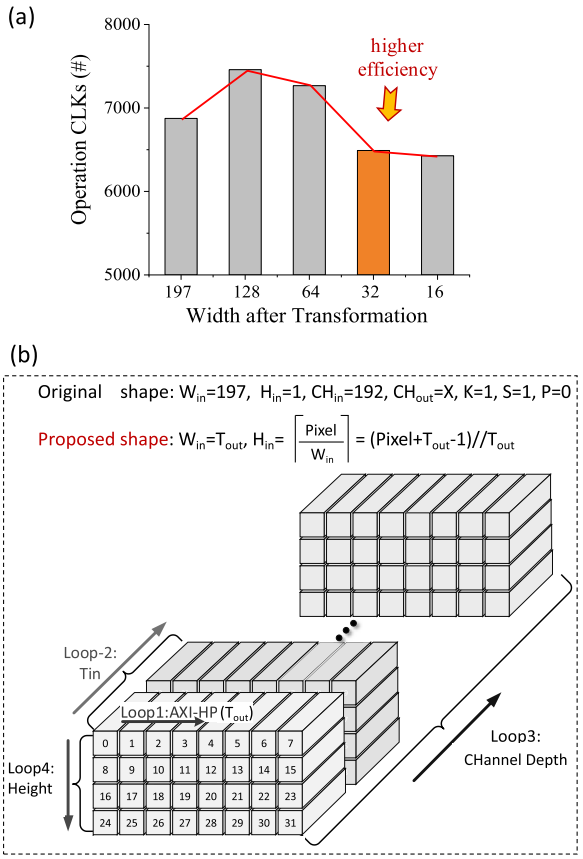

Fig. 11. (a) Efficiency of the (197,192)*(192,192) matrix multiplication at different equivalent transformation types. (b) DMA for the computation.

(Fig.10b), thus the involved registers as well as the power and area can be largely decreased. Though it usually suffers lower frequency (hundreds MHz) because of the high fan-out and coarse-grained dataflow. This work proposes group-vector systolic array (Fig.10c), which is updated from the VSA. The difference is that the group-vector transfers one row of feature data to multiple PE-rows, while the normal VSA transfers feature data to only one row. The group-vector systolic array is able to decrease the utilization of FF(Register) due to the higher parallelism. Therefore, we would like to say the proposed group vector systolic array can balance the hardware efficiency and energy efficiency, showing the best PPA (performance, power and area) result.

For the application of PE array, Yunji. Chen et.al. proposed the famous and efficient CNN accelerator several years ago [24]. They changed the convolution loop orders to make the weight data be stationary in the calculation thus increasing the weight-reuse efficiency. We adopt the above method in the acceleration of matrix computation in this work.

When accelerating ViT, more optimizations should be specifically applied. To fit the other matrix operation, we make an equivalent transformation of the matrix multiplication. As we have discussed before, the QKV matrix multiplication is a subset of the convolution operation with K=1, S=1, P=0. Take the Key matrix (denoted as K) as an example, the input activation matrix is (197, 192), the weight matrix of W_k is (192, 192), so that the output matrix will be (197, 192). In such matrix-multiplication operation, we set the input feature width

W_{in} =197, input feature height H_{in} =1, input channel depth CH_{in} =192, and output feature depth CH_{out} =192. In fact, the matrix-multiplication can be equivalently transformed into another shape with different efficiency. For the same task of computation the matrix multiplication (197,192)*(192,192), we get set the input feature width to be different ones. And the efficiency also varies since the smallest segmentation group is feature row in the design of accelerator.

Here we set the input feature width W_{in} to be variable, then the input feature height will be H_{in} = (197+W_{in} -1)/W_{in}. For example, if W_{in} is 32, the H_{in} will be 7. Note that 32*7=224 > 197, it means some pixels in the last row is meaningless, and can be filled with zero. Fig.11a shows the simulated operation clocks for the task at different Width settings. It can be seen that when W_{in} is 128 or 64, the latency cost is higher than the original 197 pixels per row. If W_{in} is set to be 64, H_{in} will be 4, then the total pixels will be 256 (=$H_{in}^{*}W_{in}$ =4∗64), which is much higher than 197. Though a smaller W_{in} leads better efficiency in the hardware design, the invalid computation cost much more operations clocks than its benefits. Such trade-off makes the matrix multiplication efficiency decreases. When W_{in} is 32, the efficiency is much higher due to the smaller task segmentation(which is dominated by pixel width). When Win is 16, the efficiency is similar to that of 32 because of the performance saturation. In consideration of the other matrix operations, the dataflow at W_{in} =T_{out} =32 is able to combined with the matrix transformation operation, which will be discussed later.

Fig.11b exhibits DMA design for the equivalent MatMUL transformation. The transformed matrix multiplication can be described with four loops. The Advanced extensible Interface (AXI) high performance burst transfer (loop-1) occupies the top priority in the computation. The feature data cube is ordered in [CH/T_{out}][H][W][T_{out}], thus loop-1 represents a continuous increment of address in its width direction, which satisfied the AXI HP protocol. The T_{in} loop has the second highest priority in the DMA transfer. This is because the matrix transpose operation need T_{in} numbers of data package. The channel depth and height loop have the lowest priority until all of the output feature pixels are finished.

C. Acceleration of Multi-Head Self-Attention

Multi-head self-attention is considered to be the key innovation for Transformer. Take the ViT-tiny model as an example (Fig.12). The shape of the Input activation is (197, 192), and weight matrix of W_q, W_k, W_v is (192, 192). After the matrix multiplication, we will get the three matrix of Query-Key-Value in (197, 192). In the multi-head self-attention mechanism, the Query-Key-Value matrix will be further reshaped according to the head numbers. If the head number is 3, the actual Query-Key-Value matrix will be a three- dimensional matrix of (3, 197, 64). Next, the context information will be calculated by $Q^{*}K^{T}$, in which Q is the Query matrix, $K^{T}$is the transpose of Key matrix. It should be noted that K is generated by the former matrix multiplication of INPUT*W_k, therefore $K^{T}$is also generated on-line and can not be pre-treated during the inference process. The attention matrix (3, 197, 197) will be performed by Softmax and finally multiplied by the Value matrix (3, 197, 64) and get the output matrix with a shape of (3, 197, 64). Since QKV are all high dimensional tensors, the above function will involve irregular and inconsecutive memory access issue. Intuitively, the complex operations involving high-dimensional tensor data are difficult to be deployed on edge.

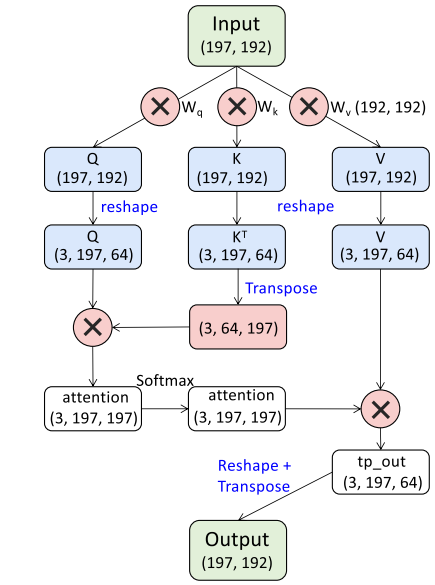

Fig. 12. Structure of multi-head self-attention block, which contains MatMUL, multi-head matrix reshape, matrix transpose and softmax.

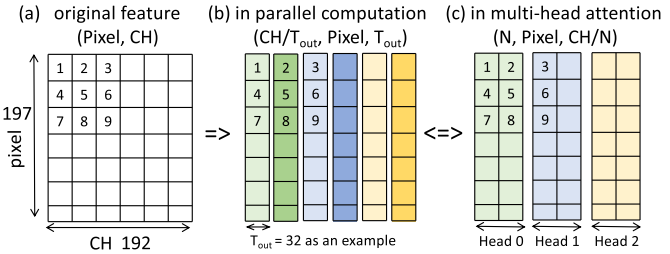

Fig. 13. Multi-head matrix reshape in the accelerator. The data structure in parallel computation and multi-head attention are exactly the same.

The multi-head self-attention mechanism contains two important key points to be figured out. One is multi-head, and the other one is self-attention. For the multi-head representation. Take the Query-Key-Value matrix as an example, if the head number is 3, the actual QKV matrix will be a three-dimensional matrix of (3, 197, 64). The above matrix can be represented as (N, Pixel, CH/N), where N is the head number, pixel is the activation pixels in each channel, and CH is the channel depth. For feature data in ViT, the original matrix is in shape of (197, 192), therefore Pixel=197, and CH=192. It can be seen that the shape of (N, Pixel, CH/N) is quite similar to that of the packaged feature (CH/T_{out}, H, W, T_{out}). Suppose the H and W can be directly shrunk into one dimension of Pixel, the feature will be (CH/T_{out}, Pixel, T_{out}). If the number of head number N equals to CH/T_{out}, these two representations will be identical. If not, the data structure during parallel computation and multi-head attention are still the same, which has been shown in Fig.13. We suggest the parallel factor of T_{out} is no larger than CH/N. Typically, we can choose T_{out} to be 16 or 32 or 64 according to the

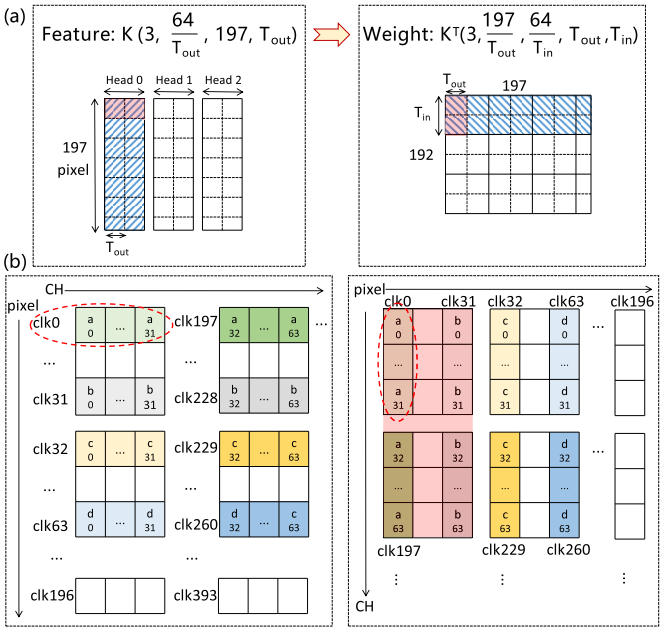

Fig. 14. Matrix transpose with the function of data package from feature (N, CH/T_{out}, Pixel, T_{out}) to weight (N, CH_{out}/T_{out}, CH_{in}/T_{in}, T_{out}, T_{in}), in which N is the head number, Pixel=CH_{out} =197, CH_{in} =192, T_{out} =32, T_{in} =64. It should be pointed that the proposed design is fully flexible and the above parameters can be largely tuned. The only constraint is $T_{out} \leq T_{in}$.

FPGA resources. In this way, the matrix reshape operation can be ignored in the multi-head attention block, since the matrix has been shaped in the convolution or matrix-multiplication computation.

Next issue is the matrix transpose for self-attention, which is more complicated and worthy of optimization research. In the classic CNN network, all of the weight parameters can be pre-treated off-line. However, in Transformer-like networks, the self-attention mechanism requires the real-time generated weight parameters, namely the transpose matrix of Key (K^T). To adapt the subsequent matrix-multiplication, the data structure of K^T matrix ought to be exactly the same as that of weight. We will start the analysis from the data structure of matrix Key (denoted as K). The original K is in shape of (197, 192). In consideration of the multi-head attention mode, the K will be reshaped as (3, 197, 64). In the hardware design, the data is actually packaged in a parallel factor of T_{out}. If T_{out} = 64, the data shape is what we want. If T_{out} is less than 64, the true data structure of K will be (3, 64/T_{out}, 197, T_{out}). For the transpose matrix of Key (denoted as K^T), the original K^T is in shape of (3, 64, 197), in which 3 is the head number, 64 is for the accumulation and 197 represents the output column numbers (output channel depth). In view of the weight parameter for matrix computation, the K^Tought to be (CH_{out}/T_{out}, CH_{in}/T_{in}, K_y, K_x, T_{out}, T_{in}) = (197/T_{out}, 64/T_{in}, 1, 1, T_{out}, T_{in}). Since the head number is 3, and kernel size (K_x = K_y) is fixed to be 1 in matrix multiplication, such that the desired K^Tcan be simplified as (3, 197/T_{out}, 64/T_{in}, T_{out}, T_{in}). Then the question will be how to effectively transpose the original feature K (3, 64/T_{out}, 197, T_{out}) to weight K^T(3, 197/T_{out}, 64/T_{in}, T_{out}, T_{in}). Fig. 14 shows our design of matrix data processor for the matrix transpose operation.

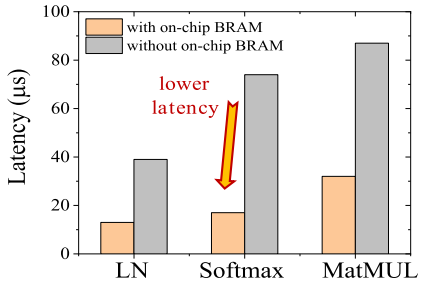

Fig. 15. Latency test of the LayerNorm, Softmax and MatMUL operation with/without the on-chip BRAM.

It should be noted that the feature matrix of K has already been packaged with a parallel factor of T_{out}, namely K(3, 64/T_{out}, 197, T_{out}), thus the transpose operation will be easier to be realized. As shown in Fig.14b left panel, the original feature data group of a0, a1, . . . , a31 are packaged. In the right panel, the transposed weight data are still packaged together such as group of a0, a1, . . . , a31. That is to say, we do not need to break down the old data package and re-construct a new data package. What we need to do is just to re-order the data package address.

If $T_{in} \neq T_{out}$, take T_{in} =64, T_{out} =32 as an example, the smallest weight data group is a rectangle package with Height=T_{in} and Width=T_{out} as shown in Fig.14b right panel. To fill such weight data group, we need to move the packaged feature data by T_{in} = 64 clocks. At clock0, we move the light-green package of A(a0∼a31). At clock31, we move the light-gray package of B(b0∼b31). From clock0 to clock31, the moved T_{out} data packages are contiguously restored and their addresses are continuously increasing. While at clock32, the address is now move to the data package of C(c0∼c31), which is not in the desired weight package. The already coming data package from A to B (total number is T_{out} =32) will have to been temporarily stored.

If T_{in} = T_{out}, the data arrangement will be much easier, because the data type from left panel to right panel are exactly the same. There is nothing to do (such as wait or re-arrange the data packages) but calculate the new address in weight data type and send them out to the next step.

D. Optimizations on Memory Data Access

There Are Three Important Points Need to Be Discussed on the Memory Access. Firstly, Different From CNN Where One Convolution Kernel Filter Can Slide Through the Entire Image With High Data Reuse Times, the Reuse in Transformer-Like Networks Are Indeed Small, Leading to Even More Serious Memory Access Bottlenecks. Secondly, All of the Operations in ViT Are Executed Layer-by-Layer, and Each Layer Desires Large Amounts of Memory Access on the Input Features and Output Features. Thirdly, Fortunately, the Feature Map Size in ViT Is Almost Fixed and Relative Small. For Example, in the ViT-Tiny Model, the Feature Is Always in Shape of (pixel, channel)=(197, 192), Namely the Total Size Is 197*192*8bit = 295.5Kb. While in the ViT-Small Model, the Feature Is Always in Shape of (pixel, channel)=(197, 384), Namely the Total Size Is 197*384*8bit = 591Kb. In Both ViT Cases, the Required on-Chip Feature Memory Cost Is

TABLE I
RESOURCE UTILIZATION ON ZCU102

ZCU102 @300MHz	LUT	FF	BRAM	DSP
CONV	65448	106610	482	1031
LayerNorm	10558	4038	9	74
Softmax	22865	21770	0	128
Gelu	10163	5992	0	32
Transpose	1373	201	4	0
On-chip Memory	421	61	128	0
AXI and DDR related	33658	29333	25	3
Total	144486 (52.7%)	168005 (30.6%)	648 (71.1%)	1268 (50.3%)

Small, Which Makes It Possible to Design Additional on-Chip BRAM on FPGA. Fig.15 Shows the Tested Results on Latency of the Several Operations. It Can Be Seen That the Accelerator Can Decrease About 60% Latency With the Help of on-Chip BRAM.

V. SIMULATION AND EXPERIMENTAL RESULTS

We have presented the design of a dedicated accelerator for ViT, which comprises a matrix computation unit, matrix transformation unit, on-chip memory (BRAM), and other components. The proposed ViT accelerator has been synthesized and implemented using Vivado 2018.1 on a ZCU102 board, which features 274K logic elements (LUT), 548K FF, 2520 DSP blocks, and 912 Block RAMs. To verify the practicality of our design, the accelerator will be integrated using AXI (Advanced eXtensible Interface) protocol and connected to the microprocessor and PL-SDRAM in the FPGA. Table I summarizes the resource utilization on the ZCU102 FPGA board for $T_{in} = 64$ and $T_{out} = 32$ combinations.

A. Acceleration of Vision Transformer

The design of the ViT accelerator includes several key modules such as the matrix computation module for performing convolution and matrix-multiplication, a matrix data processing module for handling operations such as LayerNorm and Softmax. To achieve high performance, one-DSP-for-two INT8 multiplication scheme is employed to fully utilize the available DSP resources. The well-defined pipeline dataflow in the hardware design allows for an operation frequency of up to 300MHz. The total utilization of LUT is approximately 52.7%, while the computation utilization of DSP is 50.3%. Additionally, 71.1% of the BRAM resources are allocated to the matrix-computation module and the additional on-chip memory.

The measurement flow can be described as follows. Firstly, breakdown the whole network model into several sub-tasks. For example, the first task in ViT is to execute the convolution operation for patch embedding, and the second task is to execute the concat operation for position embedding... The total task number can be about 100 in the whole ViT network. Secondly, execute the individual task one-by-one. In each sub-task, it usually contains DMA for data input, data execution, DMA for data output. To reduce the latency in each task, complex handshake protocol is used in the dataflow control. To eliminate the latency between every two tasks (for example task-A to task-B to task-C), it requires the next input feature map is exactly the previous output feature map, namely $(\text{task-A})_{out} = (\text{task-B})_{in}$, $(\text{task-B})_{out} = (\text{task-C})_{in}$, and so on. By the way, since each task needs its input and output memory access for the data transmission, it is highly recommended to group different operations in the hardware design thus decreasing the data access times. Especially for the matrix computation, the basic control flow can be described as follows. At the very beginning, all of the weight data and feature data are pre-treated and stored in off-chip memory (SDRAM). Once the computation is started, the accelerator will send instructions to fetch the feature data and weight data by the DMA (direct memory access) module. Then the data will be stored in the on-chip buffer (BRAM). Next, the computation control module will move the on-chip buffer data to the computing module (such as the systolic array). Thirdly, the temporary summation result will accumulate in the accumulation module until the corresponding $CH_{in}^{*}K_{y}^{*}K_{x}$ loops are finished. Finally, the output result will be sent out to the out-chip memory by the writing back DMA.

The accelerator is firstly tested on the ViT-small model, which contains 12 Transformer-blocks, and each block contains 6 heads, the patch_size is 16 and patches_num is 196, patches_depth = 768, encoder_dim = 384, embedding_num = patches_num + 1 = 197, hidden_dim = 4 * encoder_dim = 1536. The INT8 quantized ViT-s contains 21.4M weight parameters and shows 81.27% Top-1 accuracy. The detailed information including the input/output tensor data shapes, and the acceleration delays for each operation has been summarized in Table II. The whole ViT network can be divided into three parts, namely the embedding, transformer-block and the classification. The total latency for the whole network is only 11.14ms, corresponding to 89.76 fps, and the average throughput is 762.7 GOP per second. The power consumtion is also measured. The Idle power of the FPGA board is about ~27W, and overall power when accelerating the ViT network is 29.6W.

Table II shows the experiment result of another vision Transformer model, namely ViT-tiny. The model size of ViT-tiny is only 6.1M Bytes, which is much smaller than other models, but the Top-1 accuracy of which can reach as high as 73%. Such high efficiency makes it particularly suitable for the edge intelligence. The total latency for the whole network is only 4.077ms, corresponding to 245.3 fps, and the average throughput is 616.14 GOP per second.

The acceleration performance on ViT-s and ViT-tiny, and the individual Matrix-multiplication are always different. The primary reason is the data access. There exist two kinds of operations in this accelerator, one is free of weight parameter transmission (such as Softmax and Matrix-transpose) or needs only a few of the weights (such as LayerNorm), the other one is matrix computation that needs large number of weight parameters move from out-chip DDR to the on-chip BRAM. For the former one, as we have designed additional on-chip memory for the storage of feature data, the data access will not

TABLE II
ACCELERATION OF VIT-SMALL AND VIT-TIYN ON ZCU102

ZCU102 ViT-small	Operation		Input feature	weight	Output feature	MACs (GOP)	Delay (μs)	Throughput (GOP/s)	*Power(W) Idle:26.95*
Embed	Convolution		(224,224,3)	(16,16,3,384)	(14,14,384)	0.1156	305	379.0	*29.97*
	Concat		(14,14,384)	(1,1,384)	(197,384)	-	30	-	*27.92*
	LayerNorm		(197,384)	-	(197,384)	-	25	-	*27.72*
Block * 12	MSA head num=6	QKV	(197,384)	(384,1152)	(197,1152)	0.1743	168	1037	*29.02*
		reshape	(197,384)	-	(6,197,64)	-	0	-	
		K-transpose	(6,197,64)	-	(6,64,197)	-	22	-	*27.96*
		attn=Q*K^T	(6,197,64)	(6,64,197)	(6,197,197)	0.0298	46	647.9	*28.82*
		Softmax	(6,197,197)	-	(6,197,197)	-	102	-	*28.32*
		attn*V with shortcut	(6,197,197)	(6,197,64)	(6,197,64)	0.0298	67	444.9	*28.62*
	MLP	LayerNorm	(197,384)	-	(197,384)	-	25	-	*27.69*
		MatMul with Gelu	(197,384)	(384,1536)	(197,1536)	0.232	221	1052	*31.45*
		MatMul with shortcut	(197,1536)	(1536,384)	(197,384)	0.232	238	976.4	*29.69*
CLS	LayerNorm		(197,384)	-	(1,384)	-	25	-	*27.75*
	Full connection		(1,384)	(384,1000)	(1,1000)	0.00077	92	8.35	*28.68*
Total	-	-	-	21.4M	-	8.50	11145	762.7	*29.6*

ZCU102 ViT-tiny	Operation		Input feature	weight	Output feature	MACs (GOP)	Delay (μs)	Throughput (GOP/s)
Embed	Convolution		(224,224,3)	(16,16,3,192)	(14,14,192)	0.0578	186	310.7
	Concat		(14,14,192)	(1,1,192)	(197,192)	-	15	-
	LayerNorm		(197,192)		(197,192)	-	13	-
Block * 12	MSA head num=3	QKV	(197,192)	(192,576)	(197,576)	0.0436	44	990
		reshape	(197,192)	-	(3,197,64)	-	0	-
		K-transpose	(3,197,64)		(3,64,197)	-	11	-
		attn=Q*K^T	(3,197,64)	(3,64,197)	(3,197,197)	0.0153	23	664.8
		Softmax	(3,197,197)		(3,197,197)	-	51	-
		attn*V with shortcut	(3,197,197)	(3,197,64)	(3,197,64)	0.0149	32	465.7
	MLP	LayerNorm	(197,192)		(197,192)	-	13	-
		MatMul with Gelu	(197,192)	(192,768)	(197,768)	0.0581	57	1019.2
		MatMul with shortcut	(197,768)	(768,192)	(197,192)	0.0581	68	854.4
CLS	LayerNorm		(197,192)	-	(1,192)		13	
	Full connection		(1,192)	(192,1000)	(1,1000)	0.00038	46	8.34
Total	-	-	-	6.1M	-	2.512	4077	616.14

be the bottleneck, therefore the hardware efficiency of Softmax and LayerNorm can be indeed large and near to 100%. For the later one, the latency in data access will dominate the acceleration. For example, the weight number of ViT-tiny in QKV calculation and MatMul with Gelu (see Table II) is 192*576*8 = 864kb and 192*768*8 = 1152 kb, respectively. Meanwhile the operation delay time is 44μs and 57μs, respectively. The ration is almost the same. Overall, the hardware efficiency is rather high and close to the theoritical value of $T_{in}^{*}T_{out}^{*}Freq*2$ = 1229 GOP/s in most layers, indicating its good performance.

B. Comparison With Previous State-of-the-Art

The proposed design is also compared with related, stat-ofthe-art works. And the detailed performance comparison has been shown in Table III. Based on the INT8 quantization scheme, unified data package scheme and parallel data processing strategy, flexible on-chip and off-chip data storage management, this work shows high frequency and high throughput up to 762.7 GOP/s.

The work of [25] maps a 172 GFLOP Neural Machine Translation model with mixed-precision representation to a single FPGA board, which is the first work on implementation a real-life end-to-end NMT model to FPGA. The proposed design achieved much better performance by applying optimization techniques, such as partial on-chip weight storage, weight sharing, buffer sharing, optimized matrix-vector multiplication IPs, array partitioning, loop unrolling and pipelining. But the work is operated on low frequency of 100MHz and shows relative low throughput. Reference [26] introduces an efficient Transformer acceleration framework for FPGA application, and proposes column balanced block-wise pruning method which achieves low accuracy decay under a high pruning ratio. Besides, specialized process element for sparse matrix multiplication is designed to enable both inter block hardware parallelism and intrablock parallelism. But the sparse or pruning scheme shows relative weak generalization ability, and desires complex network training process. Reference [27] propose an algorithm-hardware co-design for the attention mechanism on FPGA. The proposed compression method effectively compresses the attention mechanism by 95%. As a result of the co-design, they developed high performance hardware accelerator achieving a run-time computing throughput of 1.87 Tops. While the practical end-to-end throughput performance is not that high, with only 190.1 GOP/s. Reference [28] propose a hardware-efficient image-adaptive token pruning framework called HeatViT for efficient yet accurate ViT acceleration on embedded FPGAs. Their work shows higher network accuracy, but the hardware efficiency is lower than ours though purning is considered. Reference [16] proposes the ViA, a novel ViT accelerator architecture based on FPGA, to execute the transformer application efficiently.

TABLE III
PERFORMANCE COMPARISON WITH PREVIOUS WORKS

	FPGA platform	Model	data type	DSP utilization	LUT utilization	Freq.	latency in MSA	Total latency	throughput	strategy
this work	ZCU102	ViT-tiny	W8A8	1268/2520 (50.3%)	144/274 (52.7%)	300 MHz	0.178 ms	4.077ms	616.1 GOP/s	Integer-ViT, group-vector systolic
		ViT-s					0.405ms	11.15ms	762.7 GOP/s	
[28] HPCA'23	ZCU102	Pruned ViT-tiny	W8	1968/2520 (78.1%)	138/274 (50.4%)	-	3.69 ms @1.74x Pruning		203 GOP/s	Pruning
		Pruned ViT-s	W8	1955/2520 (77.6%)	145/274 (52.9%)	-	9.16ms @2.27x Pruning		220.6 GOP/s	Pruning
[20] DAC'22	ZCU102	ViT-s	W8A8+ W4A8	1552/2520 (61.6%)	185/274 (67.5%)	150 MHz	-	-	907.8 GOPs	PoT quantization
[16] TCAD'22	Alveo U50	Swin T	F16	2420/5952 (40.6%)	258/872 (29.6%)	300 MHz	-	-	309.6 GOPs	optimized dataflow
[27] TECS'21	ZCU102	Multi30K	W8A8	2500/2520 (99.2%)	252/274 (91.9%)	125 MHz	-	6.8 ms	190.1 GOPs	Algorithm-hardware Co-design
[26] ISQES'21	Alveo U200	-	-	2343/6840 (34.3%)	473/1882 (25.1%)	-	0.554 ms	10.35ms	-	pruning
[25] TPDS'20	VCU118	NMT	Float32 Half16	4838/6840 (70.7%)	-	100 MHz	-	-	22 GFLOPs	mixed-precision

In data locality, they analyze the data structure in the ViT and design an appropriate partition strategy to reduce the impact of data locality in the image and improve the efficiency of computation and memory access. They obtained nearly 309.6 GOP/s computing performance in the peek on FP16 data type is still lower than ours. Reference [20] propose an FPGA-aware automatic ViT acceleration (Auto-ViT-Acc) framework for mixed-scheme ViT algorithm (low-bit with PoT quantization). It achieves higher throughput than ours, but the mixed quantized network sacrifices relative lower accuracy. Besides, the mixed-scheme quantization is difficult to train, especially for the large network.

VI. CONCLUSION

This study proposes a high-performance acceleration scheme for the Vision Transformer on FPGA, utilizing INT8 quantization and full hardware design to perform the entire ViT inference with integer arithmetic or bit-shifting. Besides, A unified data package scheme and group-vector systolic array are developed for efficiently operating the various matrix multiplication. The study also presents dynamic and flexible on-chip and off-chip data storage management strategies to reduce memory access times and improve efficiency. The proposed method is versatile and can be applied to other Transformer-like accelerators. The proposed design is deployed on the Xilinx ZCU102 FPGA platform and used to accelerate two typical iT networks as case studies. The average throughput reaches as high as 762.7 GOPs, demonstrating significant acceleration performance improvement compared to previous state-of-the-art accelerators.

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[3] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.

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

[5] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.

[6] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10674–10685.

[7] A. Radford et al., "Learning transferable visual models from natural language supervision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 8748–8763.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[10] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 10347–10357.

[11] Z. Li and Q. Gu, "I-ViT: Integer-only quantization for efficient Vision Transformer inference," 2022, *arXiv:2207.01405*.

[12] B. Li et al., "FTRANS: Energy-efficient acceleration of transformers using FPGA," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*, 2020, pp. 175–180.

[13] P. Xue, L. Pan, L. Sun, and M. Huang, "Dual-line-systolic array for high performance CNN accelerator," in *Proc. IEEE 30th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, May 2022, p. 1.

[14] H. Khan, A. Khan, Z. Khan, L. Bin Huang, K. Wang, and L. He, "NPE: An FPGA-based overlay processor for natural language processing," 2021, *arXiv:2104.06535*.
[15] P. Qi et al., "Accelerating framework of transformer by hardware design and model compression co-optimization," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.
[16] T. Wang et al., "ViA: A novel vision-transformer accelerator based on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4088–4099, Nov. 2022.
[17] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2015, pp. 161–170, doi: 10.1145/2684746.2689060.
[18] L. Yuan et al., "Tokens-to-Token ViT: Training Vision Transformers from scratch on ImageNet," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 538–547.
[19] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-BERT: Integer-only BERT quantization," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2021, pp. 5506–5518.
[20] M. Sun et al., "FPGA-aware automatic acceleration framework for Vision Transformer with mixed-scheme quantization: Late breaking results," in *Proc. 59th ACM/IEEE Design Automat. Conf.*, Jul. 2022, pp. 1394–1395.
[21] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Toronto, ON, Canada, Jun. 2017, pp. 1–12, doi: 10.1145/3079856.3080246.
[22] K. Li et al., "A precision-scalable energy-efficient bit-split-and-combination vector systolic accelerator for NAS-optimized DNNs on edge," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 730–735.
[23] M. Huang et al., "A high performance multi-bit-width booth vector systolic accelerator for NAS optimized deep learning neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 9, pp. 3619–3631, Sep. 2022, doi: 10.1109/TCSI.2022.3178474.
[24] Y. Chen et al., "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.* Washington, DC, USA: IEEE Computer Society, Dec. 2014, pp. 609–622, doi: 10.1109/MICRO.2014.58.
[25] Q. Li, X. Zhang, J. Xiong, W.-M. Hwu, and D. Chen, "Efficient methods for mapping neural machine translator on FPGAs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1866–1877, Jul. 2021.
[26] H. Peng et al., "Accelerating transformer-based deep learning models on FPGAs using column balanced block pruning," in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 142–148.
[27] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on FPGA devices," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5s, pp. 1–24, Oct. 2021.
[28] P. Dong et al., "HeatViT: Hardware-efficient adaptive token pruning for Vision Transformers," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 442–455.

Mingqiang Huang received the B.Eng. and Ph.D. degrees from the Huazhong University of Science and Technology, Wuhan, China, in 2013 and 2018, respectively. Then, he was a Research Fellow with Nanyang Technological University, Singapore, focusing on energy efficient computing and microelectronics. Since November 2019, he has been with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, as a Research Associate Professor. His current research interests include memristor in-memory computing and AI accelerator.

Junyi Luo received the B.E. degree from the School of Microelectronics, Southern University of Science and Technology, Shenzhen, China, in 2023. He is currently pursuing the M.S. degree in ECE with the University of Michigan, with a track in VLSI. His research interests include digital circuit design, neural network accelerators, and FPGA.

Chenchen Ding (Member, IEEE) received the B.S. degree from the School of Microelectornics, Southern University of Science and Technology, Shenzhen, China, in 2023. He is currently pursuing the Ph.D. degree with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong. His current research interests include machine learning, very-large-scale integration (VLSI) design, and hardware-software co-design.

Zikun Wei is currently pursuing the bachelor's degree in microelectronics science and engineering with the School of Microelectronics, Southern University of Science and Technology, Shenzhen, China. His research interests include deep learning algorithms, nerual network acceleration, and co-design of software and hardware.

Sixiao Huang is currently pursuing the Ph.D. degree in microelectronics science and engineering with the School of Microelectronics, Southern University of Science and Technology, Shenzhen, China. His research interests include digital systems design and energy-efficient neural network accelerator.

Hao Yu (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, USA, in 2007. He is currently with the Southern University of Science and Technology, Shenzhen, China. His research interests include energy-efficient data links, sensors, and analysis. He has about 282 peer-reviewed IEEE/ACM publications; nine books; one best paper award of ACM transactions; three keynote talks; three best paper award nominations for Design Automation Conference in 2006, International Conference on Computer-Aided Design in 2006, and International Conference on VLSI Design Automation in Asia and South Pacific Region in 2012; one inventor award from semiconductor research co-operation; and 20 granted patents. He is an Associate Editor of *Nature*, *Scientific Reports*, IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, *ACM Transactions on Embedded Computing Systems*, and *Microelectronics* (Elsevier); and a Technical Program Committee Member of IEEE Custom Integrated Circuits Conference, IEEE Asian Solid-State Circuits Conference, ACM-DAC, and ACM Design, Automation and Test in Europe Conference and Exhibition; and of many IEEE/ACM international journals and conferences. He is a Senior Member of ACM.