# An FPGA-based Object Detection Accelerator Architecture with Multi-channel Parallel Computation

Tianyong Ao, *Member, IAENG,* Suiao Yang, Lin Wang, Le Fu* and Yi Zhou

***Abstract*—Object detection algorithms are widely used but involve significant amounts of computations, posing challenges for adaptation to resource-constrained application scenarios, such as Unmanned Aerial Vehicles (UAVs) and Unmanned Surface Vehicles (USVs). To address these challenges, this paper proposes an object detection accelerator architecture with multi-channel parallel computation. In this architecture, the computationally intensive modules, including the convolution layer, pooling layer, and upsampling layer, are accelerated using hardware, and other modules are dealt with CPU embedded in the FPGA. The methods of pipeline design, loop expansion, data reordering, and other technologies are fully utilized to design hardware acceleration modules. A data transmission architecture is designed, incorporating multi-channel transmission along with ping-pong buffering and employing a blocking strategy for off-chip data access. Furthermore, the architecture incorporates multiple acceleration IP cores to minimize data transmission delays. Based on this architecture, the tiny-YOLOv4 model is optimized and implemented on FPGA as a hardware accelerator for object detection. The network model is enhanced by integrating the convolutional layer with normalization, and different attention mechanisms are applied to improve feature extraction, thereby reducing computational load and enhancing accuracy. The performance of the tiny-YOLOv4 FPGA-based accelerator is validated using the SIMD dataset. Experimental results demonstrate that the hardware accelerator performs exceptionally, consuming only 2.4W and surpassing existing alternatives. Such adaptability facilitates its integration into complex environments such as intelligent transportation systems.**

***Index Terms*—FPGA, Object Detection, Tiny-YOLOv4, Parallel Computing, High Energy Efficiency.**

## I. Introduction

THE object detection algorithm [1] with Convolutional Neural Network (CNN) is an important type of algorithm in the field of Artificial Intelligence (AI), and its application requirements are very wide [2]. Representative object detection algorithms include the SSD [3], Faster R-CNN [4], and YOLO [5]. Compared with the other two algorithms, YOLO algorithms have fast execution speed and a small calculation amount. One of the main idea of YOLO models is to transform the object detection problem into a regression problem. Since the complete picture of the image can be seen during the whole training and testing periods, the number of background prediction errors is significantly reduced compared with other network models. At the same time, the network maintains better generalization ability and has high stability when applied to new fields. Neural networks commonly employ CPU, GPU, ASIC, or FPGA for inference acceleration. In contrast to CPU and GPU, FPGA offers the benefits of lower power consumption and high energy efficiency [6]. Compared to ASIC, FPGA boasts advantages such as higher flexibility, low cost, and a shorter development cycle [7]. Currently, FPGA-based image recognition application scenarios are becoming more specific, including the detection and tracking of UAVs [8], small medical assistance systems [9], lightweight robots [10], etc. While detection accuracy is ensured, the emphasis in most of these scenarios is on implementing low-power and low-cost detection methods. Therefore, there is an urgent demand for high energy efficiency, low power consumption, and low-cost object detection systems. It is of tremendous research significance to use FPGA to accelerate the reasoning of the object detection model [11].

Studies on FPGA-based neural network accelerators primarily concentrate on enhancing two key components: the performance of computational units and the efficiency of data access. Regarding computing unit performance acceleration, it is mainly reflected in reducing the amount of data and accelerating modules through preprocessing. Wang [12], Hobden [13], and Ali [14] et al. used model pruning to reduce the computational complexity of the neural network and make the network obtain faster reasoning speed. However, their model is straightforward and lacks generality. Moreover, for the pruned network, the power consumption of using GPU for inference is unusually high, which is not suitable for actual scene detection. To reduce the amount of data, Nguyen [15], Liang [16], and Courbariaux [17] et al. binarized the weight data and quantified the parameters of the model to {-1, +1} or {-1, 0, +1}, to achieve the purpose of model compression and calculation acceleration. However, the precision loss caused by binarization is also apparent. T. B. PreuBer [18], Yu [19], Li [20], Jung [21] et al. quantified the weight in multiple bits to reduce the loss of data accuracy. Data quantization can reduce the computational complexity

Manuscript received January 30, 2024; revised August 7, 2024. This work was supported part by the Program for Science and the Technology Development of Henan Province (232102211011, 242102220111), the National Natural Science Foundation of China (62176088, 62303160) and the International Strategic Innovative Project of National Key Research and the Development Program of China (2023YFE0112500).

Tianyong Ao is an Associate Professor at the School of Artificial Intelligence, University of Henan, Zhengzhou Henan 450046, China. (e-mail: tyao@vip.henu.edu.cn).

Suiao Yang is a postgraduate student at the School of Artificial Intelligence, University of Henan, Zhengzhou Henan 450046, China. (e-mail: yangsuiao@henu.edu.cn).

Lin Wang is a postgraduate student at the School of Artificial Intelligence, University of Henan, Zhengzhou Henan 450046, China. (e-mail: wl_6317@163.com).

Le Fu is a lecturer at the School of Artificial Intelligence, University of Henan, Zhengzhou Henan 450046, China. (Corresponding author, e-mail: lefu@henu.edu.cn).

Yi Zhou is a Professor at the School of Artificial Intelligence, University of Henan, Zhengzhou Henan 450046, China. (e-mail: zhouyi@henu.edu.cn).

and solve the resource problem of FPGA. Data accuracy is a crucial metric for object detection. Therefore, balancing the data accuracy and the cost of FPGA resources is crucial.

The acceleration module design is mainly embodied in how to reduce the calculation delay and improve the parallelism of the calculation. To solve this problem, Zhang [22], Guan [23], Zhang [24] et al. proposed a reconfigurable design method for a CNN accelerator based on ARM+FPGA architecture. They used matrix segmentation and time division multiplexing methods to design high-performance computing units to reduce the accelerator calculation reasoning delay. Due to the lack of continuity between computation and memory access, the actual performance of the cell is not significantly improved during inferential computation. Therefore, the design of the acceleration module should consider the data transmission mode and make full use of the advantages of FPGA parallel computing to improve the parallelism of computing. Otherwise, data transmission between computing units and storage systems may become the bottleneck of the entire design architecture.

Regarding the efficiency of data access, one should optimize storage resources as much as possible and reduce access time as much as possible. At present, the commonly used optimization methods for deploying neural networks on FPGA mainly include loop unrolling [25], loop tiling [26], and loop switching [27]. Aiming at these methods, Ma [28] and Wu [29] et al. used cyclic expansion to carry out multidimensional expansion of feature graphs and other parameters, thus improving the degree of parallelism and saving the time required for calculation. However, frequent memory access is a challenging problem for neural network FPGA deployment. References [30] and [31] proposed a feature mapping storage format, which blocks data according to the rule and reads data in sequence to improve the utilization rate of locality and reduce the number of off-chip access. The present study shows that time division multiplexing is an effective method to solve the storage problem. Combined with block rules, reusing on-chip resources can improve the utilization rate of resources and reduce the number of visits outside the storage.

Through the analysis of the above problems, this paper designs a high energy efficiency and high precision hardware accelerator combined with the characteristics of FPGA hardware and deploys the mainstream network model YOLO in object detection on the accelerator. The main contributions of this paper are as follows:

(i) A low-power and energy-efficient hardware acceleration architecture is proposed in this paper. The architecture is designed to minimize computation time delays by incorporating a combination of the ping-pong buffer for data transmission and a multi-channel parallel computing framework. Furthermore, when integrated with intelligent transportation systems, this architecture proves invaluable for large-scale remote sensing data-based traffic monitoring.

(ii) To address the challenges associated with complex timing sequences and significant computation delays in existing accelerator IP cores, acceleration engines of varying sizes for convolution kernels, pooling, and sampling are introduced. These engines are devised to resolve the memory access and computation speed mismatch. Subsequently, the design space of the accelerator is discussed.

(iii) In the training stage, diverse attention mechanisms are employed to determine the weighting factors. Furthermore, convolution and Batch Normalization (BN) layers are integrated to reduce the computational load and enhance detection accuracy.

The rest of this article is organized as follows. Section II introduces the design optimization method of each hardware accelerator module. Section III deploys the Tiny-YOLOv4 model to the accelerator. Section IV summarizes and analyzes the experimental results through simulation and physical verification. Section V summarizes the complete text, analyzes it, and discusses the future work plan.

## II. Background

### A. *Tiny-YOLOv4*

Tiny-YOLOv4 is designed based on YOLOv4. Its faster detection speed and accuracy meet the requirements of practical applications, which significantly improves the possibility of deploying object detection methods on mobile devices. As shown in Fig 1, the process of Tiny-YOLOv4 detection can be divided into five steps.

(1) Input a multi-target image and divide the image into multiple grids.

(2) Inference operation by the neural network.

(3) Obtain the classification probility of the grid, as well as the candidate boxes and confidence of each grid prediction through network interence calculation .

(4) Multiply the classification probability and confidence score of each box to obtain the confidence score of each box.

(5) Output the location and category information of the detection object.

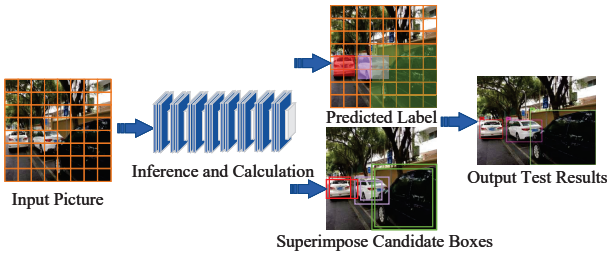


Fig. 1. The tiny-YOLOv4 inference process

The tiny-YOLOv4 network model is segmented into three components: CSPDarknet53, FPN, and YOLO_Head. CSPDarknet53 includes three BasicConv and three Resblocks, which handle fundamental convolution, activation, and pooling operations. Inspired by residual networks, this design allows gradients to flow through two separate paths, enhancing the relevance of gradient information. FPN primarily performs tensor splicing and sampling calculations, utilizing a feature pyramid network to extract feature maps at various scales and improve object detection speed. The network's output segment referred to as YOLO_Head, accommodates two sizes for detecting targets of different scales. Modularity is embraced in the computation process of the Tiny-YOLOv4 network model by consolidating the convolution layer, pooling layer, and activation function into a single module. Convolution kernels come in two distinct types, and activation operations are typically carried out once convolution calculations are completed. After computational processing,

the results are extracted from the network, encompassing object coordinates, categories, and confidential information. Subsequently, non-maximum suppression is employed to filter out redundant boxes, yielding the final detection results. A depiction of the specific network structure is presented in Fig 2.

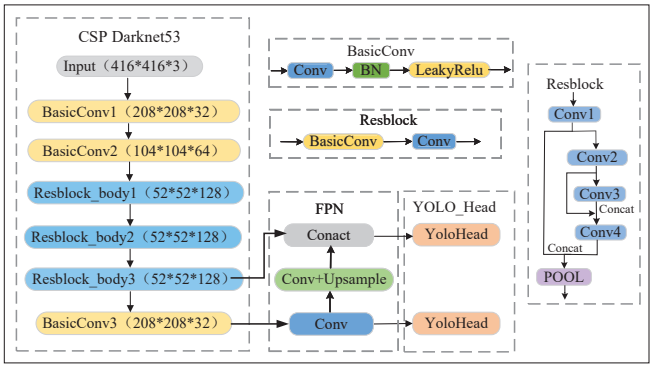


Fig. 2. Structure of Tiny-YOLOv4 network

The Tiny-YOLOv4 model can be deployed to the hardware acceleration designed above based on the network diagram. This deployment encompasses two aspects. Firstly, there is the convolution operation, which is further subdivided into two different convolution kernel sizes: 1×1 and 3×3. The focus of acceleration lies in optimizing the convolution operation. The second aspect involves lightweight computations, including tasks such as pooling and sampling layers. For the remainder of the network, the computational demands are minimal, allowing for straightforward implementation without requiring specific focus in this discussion.

### B. Object Detection Accelerator

Object detection is a technology that utilizes computers to process, analyze, and comprehend images, enabling the identification of various objects within different patterns. It made its debut during the 2012 ImageNet competition. In recent years, the most remarkable advancements in image classification, object detection, face recognition, and related domains have been rooted in deep learning. As the parallel computing performance of image recognition is hampered by hardware resources and bandwidth limitations, an increasing number of applications have turned to FPGA for image recognition. With the evolving demands of diverse scenarios and the quest for an enhanced user experience, recognition systems are subject to increasingly stringent requirements. Consequently, much of the research effort focuses on deploying image recognition algorithms on FPGAs to evaluate the proposed acceleration schemes and architectures. Given the finite resources of FPGAs, completing all neural network calculations in a single sweep is unfeasible. Instead, a time-division multiplexing method is employed for computations. The overarching approach involves transmitting images from DDR to the FPGA chip cache, storing pertinent weight information on the chip, and subsequently employing a specially designed acceleration engine to execute the relevant computations.

The neural network model is too extensive to fit entirely within on-chip memory. To address this challenge, loop tiling is employed, dividing data into blocks that can fit in on-chip memory. The primary objective of this technique is to allocate tiling sizes in a manner that optimizes data locality for convolutions and minimizes data transfers between external memories. Ideally, each input and weight is transferred to the on-chip buffer only once from external memory.

Sacrificing small precision to enhance performance is a common approach in FPGA implementation. The two most prevalent strategies involve reducing precision and the number of operations. During training, data is represented in a floating-point format. During inference, data can be converted to a fixed-point format, typically 8 or 16 bits, in order to decrease storage demands, hardware utilization, and power consumption.

### C. Parallelism and Pipeline of FPGA

The primary advantage of FPGA compared to GPU lies in its latency performance. It has the capability to output computation results for multiple channels in a single clock cycle, thereby significantly boosting throughput. FPGA's parallel processing hinges on precise clock construction within digital circuits. This clock consistently triggers various modules, propelling data progression according to predetermined objectives. There are two primary methods of parallel processing based on FPGA: data parallelism and pipeline parallelism.

Data parallelism shines in the data transmission process, facilitating computed data input into the computing engine through multiple channels. In interactions with external storage, FPGA can leverage multiple external interfaces for data transmission, effectively expanding the transmission bandwidth. In the context of image processing, FPGA typically manages multi-dimensional input feature maps and weights. To augment input and output parallelism, dimensionality reduction is achieved through loop unrolling. However, caution is warranted, as excessive unrolling may lead to increased storage consumption and area sacrifice. Thus, employing loop unrolling for moderate data parallelism necessitates logical sorting and parallel transmission based on specified rules. A proposed feature map storage format organizes data into blocks, enhancing locality utilization and increasing data transmission parallelism.

Pipeline parallelism forms the bedrock of hardware design, breaking down substantial tasks into smaller, sequentially executed operations. FPGA segments the combinational logic system, introducing registers between segments to cache intermediate results. This approach decomposes complex operations into smaller, quickly completed tasks, thereby boosting clock frequency. Pipeline parallelism is most effective when employed with as many tasks as possible, considering emptying and filling times.

In our design, loop unrolling is initially used to expand both input and output channels, enhancing data input and output parallelism with minimal resource consumption. FPGA interfaces with external storage devices through multiple external interfaces, employing pipelines for data transmission and computation. This concurrent approach significantly increases throughput across multiple tasks. It is important to note that the emphasis on loop unrolling and pipelining in this paper lies in their strategic application within our

proposed framework, addressing specific challenges and optimizing for efficiency.

### D. Convolutions of Different Sizes

Most neural networks consist of convolutional layers, pooling layers, and fully connected layers. Among these, the convolutional layer exhibits the highest complexity, resulting in the longest processing time. This article primarily focuses on accelerating the convolution layer, with the majority of current convolutions being either 1×1 or 3×3. A 1×1 convolution does not alter the height and width of the image, its primary effect lies in dimensionality augmentation or reduction. The operation of a 1×1 convolution is simpler than that of a 3×3 convolution due to its smaller kernel size. As depicted in Fig 3, it involves only a straightforward point multiplication operation within the convolution kernel, without performing a full matrix multiplication.

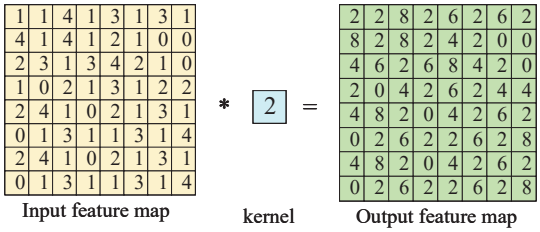


Fig. 3. 1×1 convolution calculation process

The operation involving a 3×3 convolution kernel is intricate. Within this operation, both the convolution and its interior components require multiplication and addition, as shown in Fig 4. This process is equivalent to expanding the receptive field area. As a result, the computational complexity and data transmission during the calculation process is significantly higher compared to a 1×1 convolution.

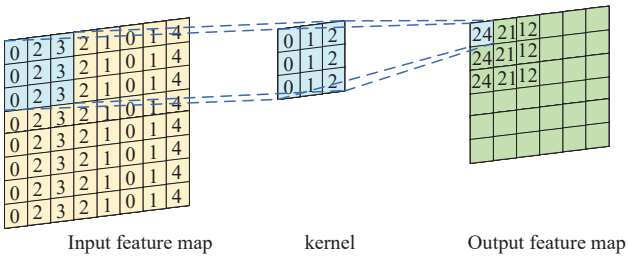


Fig. 4. 3×3 convolution calculation process

At present, most hardware acceleration methods employ 3×3 convolution instead of 1×1 convolution to minimize hardware resource usage. However, due to the differing computational loads between these two convolution sizes, data transmission and computation processes exhibit significant disparities, potentially resulting in a speed mismatch between the two stages. Consequently, this paper introduces specialized acceleration engines designed for convolutions of varying sizes. Furthermore, it should be noted that the design scope of this paper is not limited to the YOLO model, and the proposed method is equally applicable and universal in the deployment of other models.

## III. System Architecture

### A. Overview of Accelerator Architecture

The accelerator adopts an ARM+FPGA architecture to accelerate neural network inference, comprising the Processing System (PS) and Programmable Logic (PL). The PS utilizes an ARM processor for task scheduling and data loading allocation. Meanwhile, the PL leverages the parallel computing capabilities of the FPGA to execute intricate computations such as convolution, pooling, and sampling. The structure of the system is shown in Fig 5.

The PS consists of the Core Processing Unit (CPU) and the data control module, primarily responsible for preprocessing images and allocating them to the PL for reasoning and computation. This paper primarily focuses on the PL, which encompasses two convolution acceleration modules of different sizes, a pooling module, and a sampling module. To enhance data transmission speed, multiple input and output buffers have been designed. We employ a ping-pong operation to facilitate parallel calculations across multiple channels. Data interaction between the PS and PL is achieved through the AXI bus. Exploiting the high-speed data processing characteristics of the HP interface of the AXI bus, we have designed five interconnection modules in a master-slave configuration. The PL, functioning as the host, accesses off-chip storage, while the PS, acting as the slave, reads data from external storage based on address information and transfers it to on-chip cache buffers via four HP interfaces. Subsequently, the accelerator calls the Convolution IP and Maxpool & Upsample IP, following the neural network's calculation process. After the IP core computations have been accelerated, the results are written to the output buffer. The accelerator workflow can be divided into four steps:

Step 1: Image preprocessing, preparation, and the transmission of relevant information to the PL side.

Step 2: The PL engages in data interaction with the PS and loads off-chip data into the on-chip cache.

Step 3: The accelerated IP on the PL side conducts inference calculations and delivers the results to the output cache.

Step 4: The PL side writes the results back to the PS side via the HP interface and subsequently outputs the results.

### B. Multi-Channel Data Transmission

The actual bandwidth of DRAM exceeds that of a single interface. To address the memory optimization challenge in the data transmission process, a solution is proposed, which adopts a multi-channel DMA parallel data interaction mode. A pipeline buffer is also implemented to enhance the internal storage structure of the accelerator. The read and write channels on the AXI bus operate independently, and the AXI master interface module generated by Vivado HLS resembles a DMA structure. When reading data from off-chip storage, a 4-channel DMA is employed to retrieve the input feature maps and weight information. The data is initially stored in buffer A and subsequently in buffer B during the following clock cycle. Each channel has a bit width of 32 bits, effectively quadrupling the interface's bit width during parallel channel processing. This leads to improved data reading parallelism and throughput rate, as depicted in Fig 7.

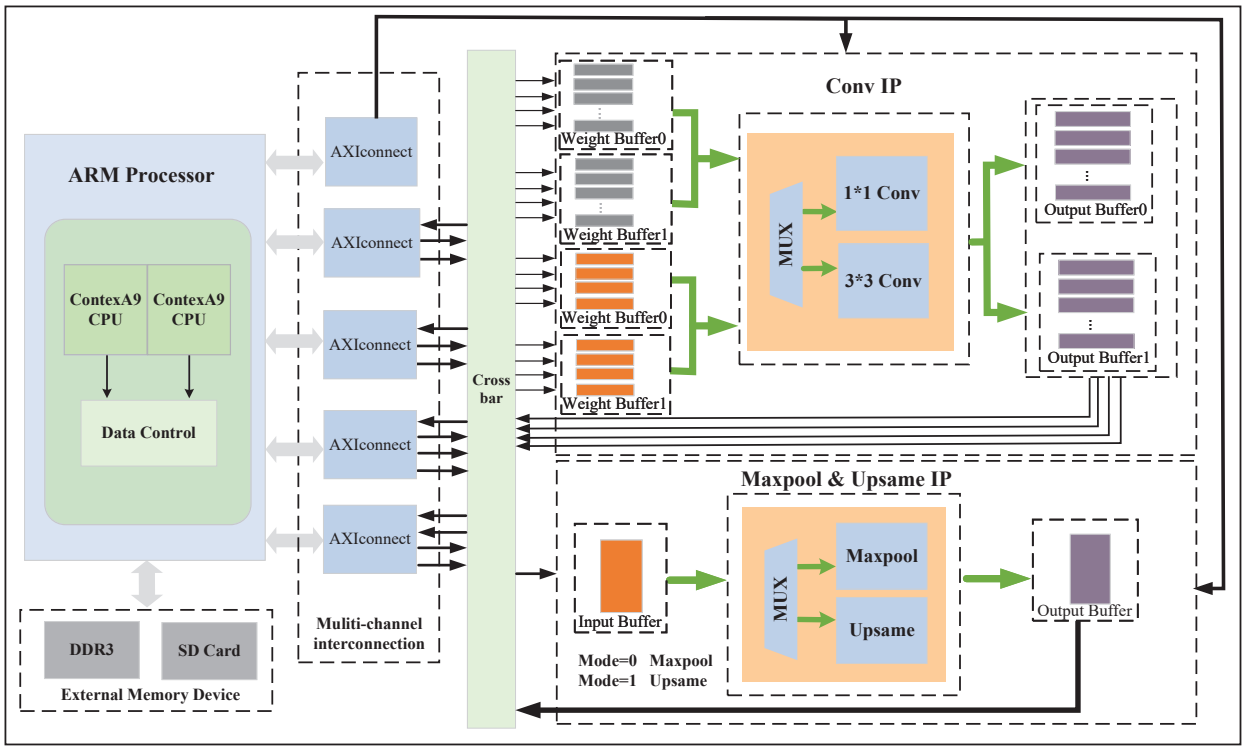


Fig. 5. Overall system design architecture diagram

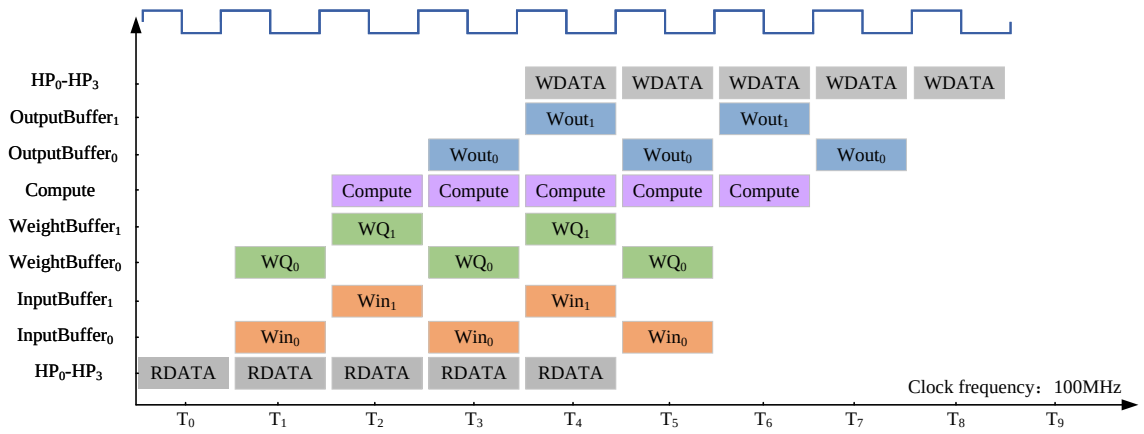


Fig. 6. Space-time diagram of convolution calculation

The space-time diagram of the entire convolution calculation is depicted in Fig 6. Initially, the PL retrieves data from external storage via the HP interface on the AXI bus, specifically the weights and input feature maps. Subsequently, this relevant data is loaded into on-chip buffers. In the design of the on-chip storage, a ping-pong buffer strategy is implemented, featuring two weight buffers, two input buffers, and two output buffers, respectively. The on-chip buffer structure is based on the ping-pong buffer design, with each buffer comprising a varying number of banks. The number of banks in the input buffer correspond to the parallel number of input channels $T_n$, while the number of banks in the output buffer align with the parallel number of output channels.

The data reading process primarily involves two aspects: the retrieval of the input feature map and the retrieval of weight data. In our design, multi-channel transmission has been implemented to enable concurrent execution of these two parts. However, the pooling module solely necessitates the transmission of the input feature map, as it does not consider the weights. The entire calculation process can be divided into the aforementioned five steps. The execution times for these five steps are denoted as $T_1$, $T_2$, $T_3$, $T_4$ and $T_5$, and the total number of tasks to be executed is represented as 'times.' In the absence of pipelining, the total time required for the entire computation flow is calculated as formula (1).

$$T_{sum} = N \times (T_1 + T_2 + T_3 + T_4 + T_5) \tag{1}$$

After implementing the ping-pong buffer, data is efficiently read from the DRAM and stored in buffer A. Simultaneously, the data in buffer A is transferred to the computing unit for computation. At the same time, feature map cache B

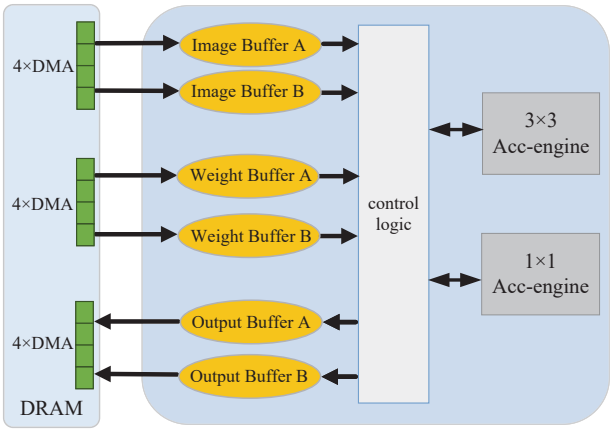


Fig. 7. Diagram of ping-pong buffering and multi-channel transmission

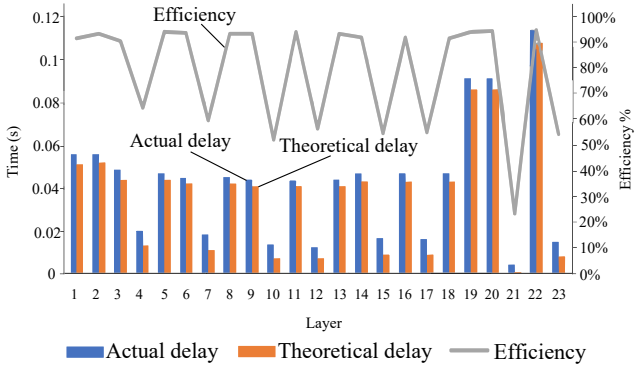


Fig. 8. Accelerator efficiency evaluation for a single IP core

accesses data from the DRAM, allowing for concurrent data retrieval and computation. This overlap in data processing optimizes access time. Furthermore, the total duration of the entire calculation is contingent on the longest part of the process. Following the principle of pipeline calculation, the overall time required for the entire calculation process can be summarized as formula (2).

$$\begin{aligned} T_{pipeline} = (N-8) \times MAX(T_1, T_2, T_3, T_4, T_5) \\ + MAX(T_1, T_2, T_3, T_4) \\ + MAX(T_1, T_2, T_3) \\ + MAX(T_1, T_2) \\ + MAX(T_2, T_3, T_4, T_5) \\ + MAX(T_3, T_4, T_5) \\ + MAX(T_4, T_5) + T_1 + T_5 \end{aligned} \tag{2}$$

Based on the inequality $T_{pipeline} \leq T_{sum}$, the optimization strategy outlined in this section effectively enhances data transfer efficiency and reduces latency.

### C. Convolution Acceleration Engine with Different Sizes

Most of the computational time in neural network models is concentrated within the convolution layer, which is our primary focus for acceleration. Due to limited FPGA resources and the sheer size of neural network models, simultaneous computation of all calculations is unfeasible. Consequently, a circular chunking strategy is employed for off-chip data reading. Circular partitioning enables the division of matrix loops into smaller modules, facilitating data reuse on-chip and minimizing memory access times.

Neural networks typically feature two types of convolution kernel sizes. The 1×1 convolution operation is replaced with the 3×3 convolution operation, and the accelerator's efficiency in the inference calculation process of the convolution layer is assessed, as illustrated in Fig 8. Notably, the less efficient layers are predominantly found in the 4th, 7th, 10th, and 12th convolution layers with a 1×1 convolution kernel size. In these instances, the layer's processing time is dictated by data transmission, with computation time falling short of data transfer time. Consequently, the computing unit must pause and await data retrieval, leading to longer actual computation times. To address this mismatch between data storage and module calculation speed, we have devised dedicated acceleration engines.

Calculation efficiency is defined as the ratio of theoretical calculation time to actual calculation time. Certain variables are defined and presented in Table I.

The accelerator employs a block strategy for reading the data from off-chip memory to input buffers. The data involves input feature maps and their weight parameters. The selection of tile size significantly influences performance, and our design is bound by the following constraints:

$$\begin{cases} 0 < T_m \times T_n < (PEs) \\ 0 < T_m \leq T_{out} \\ 0 < T_n < T_{in} \\ 0 < T_r < R_{in} \\ 0 < T_c < C_{in} \end{cases} \tag{3}$$

Following blocking rules, $T_m$ is divided into $T_{in}/T_n$ blocks in the input channel, and $T_{out}$ is divided into $T_{out}/T_m$ blocks in the output channel direction. Each original input feature map of size $R_{in} \times C_{in}$ on each channel is partitioned into $(R_{in}/r_{in}) \times (C_{in}/c_{in})$ blocks. Similarly, the output feature map $C_{out} \times R_{out}$ on the output channel is divided into $(R_{out}/r_{out}) \times (C_{out}/c_{out})$ blocks. In summary, for each layer, the convolution operation necessitates reading the convolution kernel from off-chip $(T_{in}/T_n) \times (T_{out}/T_m) \times (R_{out}/r_{out}) \times (C_{out}/c_{out})$ times, reading the feature map from off-chip $(T_{in}/T_n) \times (R_{in}/r_{in}) \times (C_{in}/c_{in})$ times, and writing back to the off-chip storage system from the output cache $(T_{out}/T_m) \times (R_{out}/r_{out}) \times (C_{out}/c_{out})$ times. The block diagram is shown in Fig 9.

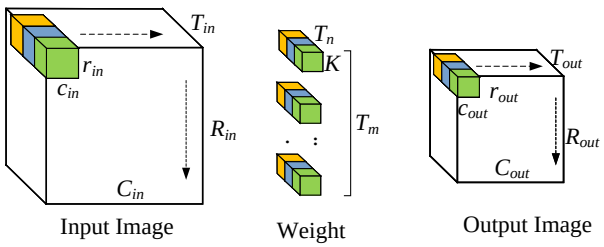


Fig. 9. Block diagram of the convolution calculation

For the 3×3 convolution kernel, the input feature map is expanded in two dimensions: the input and the output channel. Additionally, multiple parallel multiplication and addition units have been incorporated to facilitate the convolution operation. During the data calculation process,

TABLE I
TABLE OF RELATED PARAMETERS

| Symbol | Meaning |
|---|---|
| $R_{in}$ / $C_{in}$ / $R_{out}$ / $R_{out}$ | Length/width of input/output picture |
| $T_{in}$ / $T_{out}$ | Number of channels to input/output images |
| $T_n$ / $T_m$ | The parallelism between the input and output of the accelerator |
| $r_{in}$ / $r_{out}$ / $c_{in}$ / $c_{out}$ | Length /Width of input/output picture each time |
| K | Convolution kernel size |

the weight data, input pixels, and output pixels are partitioned based on the input and output channels, effectively reducing the multi-dimensional array to a two-dimensional array. Subsequently, the pixel blocks are sequentially fed into the acceleration engine for computation. To enhance FPGA's calculation speed, pipelining is employed on the output channel, allowing the results of 32 channels to be output to the buffer in a single cycle. Our design consumes approximately 128 DSP resources in this system.

During each clock cycle, the convolution module retrieves pixels from the input buffer while simultaneously fetching weight parameters from the corresponding position in the weight buffer. It employs $T_n \times T_m$ parallel multiplication units to perform multiplication calculations and subsequently add the product results in pairs using $T_m$ addition trees. Each computing unit is independent of the others, and the pipelining work cycle reflects the advantages of FPGA parallel data processing. Once all calculations are finished, the results are written to the output buffer. As shown in Fig 10, illustrating the schematic diagram of the 3×3 convolution operation.

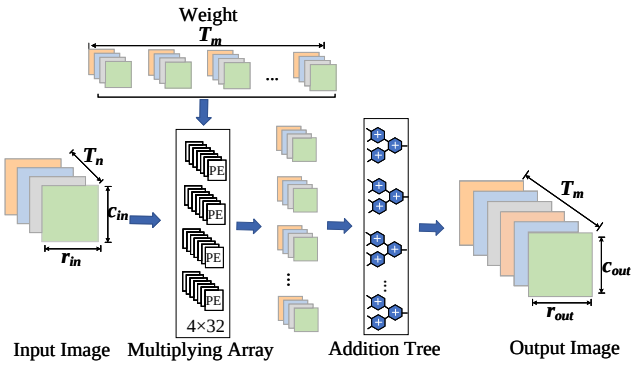


Fig. 10. 3×3 Schematic diagram of the convolution operation

The 1×1 convolution involves significantly fewer calculations compared to the 3×3 convolution. It essentially involves a dot product operation, which is amenable to a block matrix multiplication design. Similarly, the weight data, input, and output pixels are segmented into two-dimensional arrays, and pipeline processing is implemented on the output channel. In our design, each layer consists of 4 input channels and 8 output channels. The input image and weights are used for point multiplication, as depicted in Fig 11.

The convolution layer's computation is the primary focus of this acceleration system. It has been observed that the accelerator's computational efficiency depends on the clock frequency and the values of $T_m$ and $T_n$. Convolution consists of two parts: weight calculation and input feature mapping. The total amount of calculation is the sum of these two

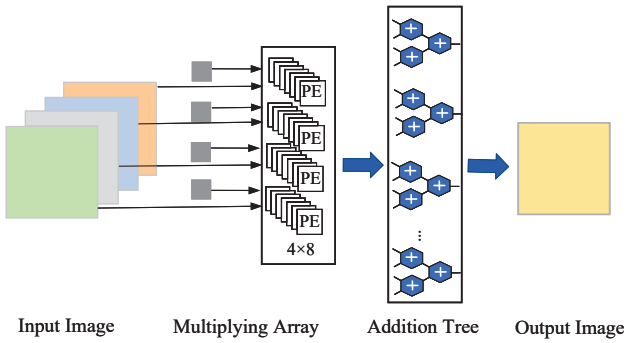


Fig. 11. 1×1 Schematic diagram of the convolution operation

parts, which are calculated in the same way, so they can be expressed by the following formula (4).

$$\mathrm{E} = 2 \times (r_{in} \times c_{in} \times T_n \times T_m \times K \times K) \tag{4}$$

Due to the inherent characteristics of the Tiny-YOLOv4 network, the minimum input channel value is constrained to 13, which we shall refer to as $T_{in}$. The value of $T_m$ is determined based on our input parallelism. With four HP interfaces on the AXI bus, four banks in each buffer of the input cache have been configured. The calculation engine processes four channels in parallel during each cycle. The network's intrinsic attributes also dictate the value of $T_m$. The network specifies a minimum of 32 output channels, and we adhere to this value. In principle, the hardware design parameters are optimized to the fullest extent allowed by Tiny-YOLOv4. However, owing to constraints posed by FPGA accelerator resources, 89% of the resources have been allocated to the convolution operation, while the remaining resources are reserved for other operations.

### *D. Other Acceleration Modules*

The pooling layer in a neural network typically employs maximum pooling with a step size of 2. Unlike the convolution layer, the pooling layer primarily relies on comparators. The number of comparators is determined by the input parallelism, denoted as $T_n$. In each clock cycle, pixel values are read from the input buffer. After conducting $S \times S$ comparisons, the maximum value is written to the output buffer. Once this layer's calculations are complete, the next layer proceeds. To enhance efficiency, parallel calculations across multiple channels are implemented by cyclically expanding the input channel. You can see the pooling circuit in Fig 12.

The sampling function serves to magnify the image's finer details. In our hardware design, this function is implemented as a multiplexer. This selector takes a single pixel as input

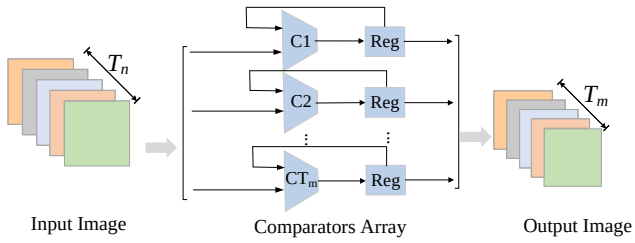


Fig. 12. Schematic diagram of the pooling operation

and creates four copies of the pixel, distributing them across four separate buffers. To enhance computational speed, a pipeline approach is employed to increase computational parallelism with an input parallelism of 4. Within the FPGA, multiple selectors are generated, and the specific structure can be referred to in Fig 13.

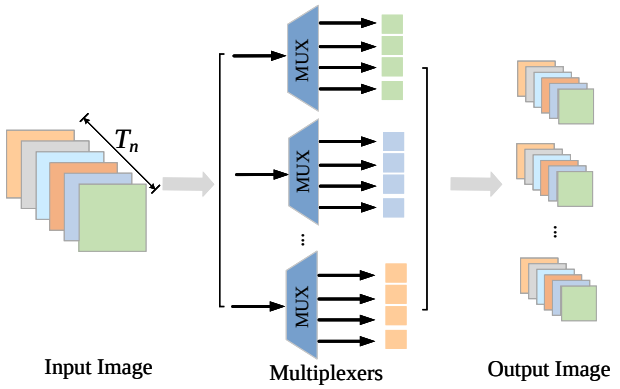


Fig. 13. Schematic diagram of the sampling operation

## IV. Deployment of the Model

### A. Convolution Layer Network Fusion

In the training of neural network models, the BN layer is typically positioned after the convolution layer to expedite network convergence and mitigate overfitting. BN normalizes the data, effectively addressing issues related to gradient vanishing and explosion. However, during network forward inference, the presence of additional calculation layers can impact model performance and consume more memory or video memory space. To enhance the speed of forward inference, it becomes necessary to consolidate BN layer parameters into the convolution layer. We achieve this integration by incorporating the convolution and BN layers into the weights, utilizing a specific calculation formula to generate a bias parameter. The original calculation process is as follows:

$$y_i = \gamma \frac{(wx_i + b) - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta \tag{5}$$

Where $\mu$ is the mean, $\sigma$ is the variance, and $\gamma$ is the scaling factor. These values are obtained through network training and are fixed in the reasoning process. Let us put $\alpha = \frac{\gamma}{\sqrt{\sigma^2+\varepsilon}}$ into the above formula $W_{new} = w \times \alpha$, The new bias $y_i = x_i \times \omega \times \alpha + \alpha \times (b - \mu) \times \beta$. And then we get the new weights $\beta_{new} = \alpha(b - \mu) + \beta$. According to the analysis in Table II, a convolution requires at least four multiplications and additions before the fusion. However, after fusion, only one multiplication and one addition are needed, reducing computation and memory retrieval times.

TABLE II
Comparison of Parameter Fusion Performance

| | Before fusion | After fusion | Promotion | Error |
|---|---|---|---|---|
| Time(s) | 0.5318 | 0.5103 | 4.2 | 5.206e-10 |

### B. Network Training with Different Attention Mechanisms

Neural network computation involves two phases: training and inference. Due to FPGA resource limitations, the CPU is utilized for network training, with essential data like weights and biases necessary for calculations being acquired. The introduction of attention mechanisms during network training enhances feature extraction, allowing the model to better capture information from the training set and ultimately achieve higher accuracy. Different attention mechanisms are explored to extract essential features for improved model precision.

One such mechanism is SENet, which employs channel attention to assign feature map weights. This mechanism optimizes feature maps by assigning higher weights to significant features and lower weights to less influential ones, directing the network's focus on critical information. While SENet performs exceptionally well on specific datasets, its universality is limited, and the surplus parameters may affect model speed.

The CBAM module extrapolates the attention map along two independent dimensions and then multiplies the attention map with the input feature map for adaptive feature optimization. The key advantage of the attention mechanism is its ability to focus on relevant information, eliminating irrelevant details and establishing direct input-output dependencies without recycling, thus enhancing parallelism.

Comparatively, the ECA attention mechanism stands out for its simplicity, computational efficiency, and low parameter count while delivering substantial performance improvements. It enhances models without adding complexity, making it a practical choice for lightweight network inference compared to existing methods.

The above analysis employed different attention mechanisms to derive network weights, and their impact on our proposed model was evaluated through experiments. This comparative analysis identified the weight data with the highest accuracy and was subsequently used for FPGA accelerator computations.

### C. Low-Bit Quantization

Convolution operations in neural networks rely heavily on multipliers and adders. Resource consumption is intrinsically tied to data accuracy. During the training of the YOLO model, the 32-bit floating-point data type is employed. Binarization and 8-bit fixed-point quantization compromise data accuracy. Hence, our initial consideration is 16-bit fixed-point quantization. Experimental results in Table III showcase the resource and power consumption for DSP, FF, and LUT in FPGA, which vary with different data accuracy.

TABLE III
COMPARISON STATISTICS OF RESOURCES AND POWER CONSUMPTION

| | DSP | FF | LUT | Power(PJ) |
|---|---|---|---|---|
| Adder(float32) | 2 | 354 | 231 | 0.9 |
| Adder(fixed16) | 0 | 96 | 52 | 0.05 |
| Multiplier(float32) | 3 | 291 | 144 | 3.7 |
| Multiplier(fixed16) | 1 | 96 | 99 | 0.9 |

Experimental results demonstrate that, when compared to the use of 32-bit floating-point data, employing 16-bit fixed-point data significantly reduces resource and energy consumption. The quantized value for 16-bit fixed-point data can be expressed using the formula (6).

$$V_{fixed} = \sum_{\mathrm{i}=0}^{15} B_i \times 2^{-fl} \times 2^i, B_i \in \{0,1\} \tag{6}$$

The theoretical FPGA resource consumption was assessed following fixed-point quantization. Formulas (7), (8), and (9) represent the respective theoretical consumption values for DSP, FF, and LUT in the hardware architecture design.

$$C_{DSP} = T_m \times T_n \times (\delta_{DSP} + \varepsilon_{DSP}) \tag{7}$$

$$C_{FF} = T_m \times T_n \times (\delta_{FF} + \varepsilon_{FF}) + T_m \times \gamma_{FF} \tag{8}$$

$$C_{LUT} = T_m \times T_n \times (\delta_{LUT} + \varepsilon_{LUT}) + T_m \times \gamma_{LUT} \tag{9}$$

Where $C_{DSP}$, $C_{FF}$, and $C_{LUT}$ represent the consumption of DSP, FF, and LUT resources during convolution operations. Meanwhile, $\delta$, $\varepsilon$, and $\gamma$ reflect resource consumption related to various data precision types. Notably, $\gamma$ is directly linked to the number of output channels. Our primary concern is the resource consumption involved in adding bias after multiplication and addition. A comparison with Table III highlights the substantial reduction in resources and power consumption achieved by using 16-bit fixed-point data types in contrast to 32-bit floating-point data.

## V. EXPERIMENTAL RESULTS

### A. Experimental Setup

Characteristic weight values were extracted by different attention mechanisms, and the optimal weights were selected through a comparative analysis. Subsequently, these weights were quantized into 16-bit fixed-point data types. Finally, these quantized weights were integrated with the proposed FPGA hardware accelerator. The hardware acceleration system utilized a Xilinx PYNQ-Z2 board featuring a Cortex-A9 chip, encompassing PL and the embedded processor PS. The hardware core module was designed using Vivado HLS 2019.2, and synthesis and layout were performed with Vivado 2019.2. For system validation, an extensive series of experiments was conducted to ensure system stability and accuracy. A comprehensive evaluation of the accelerator was conducted employing the SIMD dataset [32], considering practical application scenarios.

### B. Comparison of Training Accuracy

During the training phase, the SEnet, CBAM, and ECA attention mechanisms were incorporated individually to boost network accuracy. The SIMD dataset, comprising various vehicle types such as cars, planes, ships, and ten other standard modes of transportation, was utilized in our experiments. After the network training was completed, the optimal weights were determined. Then, various samples were evaluated, and the accuracy comparison under different attention mechanisms is shown in Fig 14.

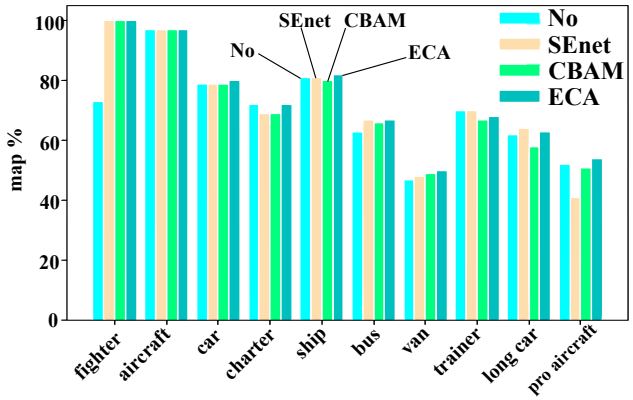


Fig. 14. Precision comparison of different attention mechanisms

The figure above clearly shows that the ECA attention mechanism consistently outperforms other attention mechanisms in terms of accuracy. This view was better illustrated by evaluating our training model under different Average Precision (AP) values, as shown in Table IV.

TABLE IV
PRECISION COMPARISON UNDER DIFFERENT AP VALUES

| | NO | CBAM | SEnet | ECA |
|---|---|---|---|---|
| Ap30 | 71.7 | 74.1 | 73.3 | 74.4 |
| Ap50 | 69.6 | 71.9 | 71.6 | 72.9 |
| Ap75 | 44.3 | 46.4 | 40.9 | 47.9 |

The above clearly shows that the combination of attention mechanisms improves the accuracy of our model, and ECA produces excellent results, achieving higher accuracy than other mechanisms. Subsequently, the best weight model was selected, and its weight data were partitioned according to each module described in Fig 2. The concept of modularity is fully reflected in the partitioning process, which alleviates multiple weight interactions in subsequent FPGA deployment and reduces the number of times the FPGA accesses external storage.

### C. Deployment Result Analysis

After determining resource allocation, the corresponding design configuration is presented in the following Table V. To optimize FPGA resource utilization prior to deployment, 16-bit fixed-point quantization of weights was conducted. Table V details the utilization of LUT, FF, BRAM, DSP, and LUTRAM. Analysis reveals that LUT, BRAM, and DSP experience high consumption rates. This is attributed to the storage of output cache data within on-chip storage, leading

to significant BRAM resource utilization. In the convolution operation, parallel processing on the output channel, along with the use of multiple multipliers, results in high DSP resource occupancy.

TABLE V
FPGA RESOURCE CONSUMPTION

| Resource | Utilization | Available | Utilization Rate(%) |
|---|---|---|---|
| LUT | 42122 | 53200 | 79.18 |
| BLUTRAM | 7414 | 17400 | 42.61 |
| FF | 47709 | 106400 | 44.84 |
| BRAM | 96 | 140 | 68.57 |
| DSP | 220 | 220 | 100 |

The hardware performance was evaluated using throughput as the criterion, which was calculated as the total number of operations divided by the execution time. Total operations serve as an indirect measure of network complexity. In the FPGA implementation of this design, the quantized virtual network entails a total of 8 GOP operations. For input feature maps sized at $416 \times 416 \times 3$, the Xilinx PYNQ-Z2 FPGA yields output results from the last layer in 300ms. GOPS stands for giga-operations per second. A remarkable overall throughput of 26 GOPS for the quantized virtual network is achieved at a 100MHz FPGA clock frequency through our design. While the PYNQ-Z2 board boasts 220 DSP resources and a 100 MHz clock frequency, practical performance falls short of the theoretical maximum of 44 GOPS due to bandwidth and clock limitations. The throughput and complexity of each convolution layer are shown in Fig 15.

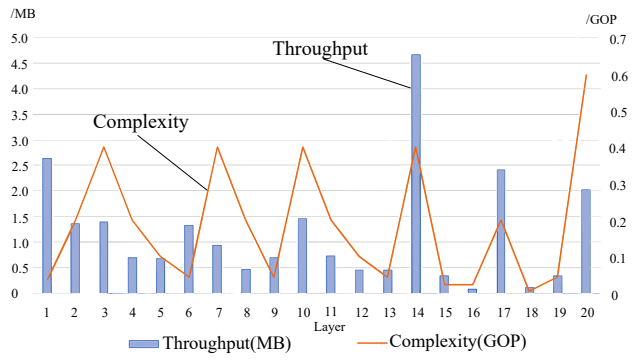


Fig. 15. Model throughput and complexity

Based on our analysis, the computational load of convolution layers like 6, 10, and 14 in the network was relatively low, with a convolution kernel size of 1×1. To accelerate these layers, a specialized convolution acceleration engine was employed. Conversely, other convolution layers have a larger size of 3×3, demanding more extensive internal computations due to their higher computational complexity. Therefore, another engine is used for acceleration, and the number of resources is four times that of the acceleration engine so that the calculation speed can be improved.

### D. Contrast with Previous Work

Table VI compares this design with previous efforts in YOLO hardware acceleration. As there was no dedicated YOLOv4 accelerator, comparisons in our experiments were made with other YOLO models. The design power consumption of this paper was only 2.4W, which was the lowest power consumption among different types of FPGAs, leading to a dramatic reduction in system power consumption. Moreover, the highest energy efficiency ratio was achieved by our design method compared to other accelerators.

Table VII presents a comparison of various types of accelerators beyond object recognition accelerators. The comparison results demonstrate that the design approach proposed in this paper offers significant accuracy and energy efficiency advantages with low power consumption and high energy efficiency in calculations.

### E. Detection Results in Actual Scenarios

To assess the stability of the hardware acceleration system, images captured by satellites were collected and subjected to detection using the system developed in this paper. The selected scenes encompass streets, docks, parking lots, airports, and other complex environments. The detection results are depicted in Fig 16.

As can be seen from the figure, the hardware accelerator we designed maintains high accuracy in complex scenarios, particularly in the field of intelligent transportation, which demands strict power consumption control. Consequently, the hardware accelerator and deployment method presented in this paper upholds high accuracy while significantly reducing the system's power consumption.

## VI. Discussion

In our research, we implemented a new type of hardware accelerator based on FPGA, designed multiple IP cores, and finally used the YOLO network model for verification and application in traffic monitoring scenarios. Our approach represents a departure from previous methodologies, encompassing advancements in hardware architecture design and network model optimization. Notably, the accelerator demonstrates superior power efficiency and energy ratio compared to CPU and GPU alternatives, while achieving higher detection accuracy. Future endeavors will concentrate on enhancing the computing performance of the proposed architecture to support real-time detection requirements. This will leverage more resources to enhance hardware capabilities and maximize on-chip cache utilization to minimize reliance on off-chip caches, thereby enhancing accelerator speed.

## VII. Conclusion

To address the issues of high power consumption and low energy efficiency in object detection on current hardware devices, a hardware accelerator based on FPGA architecture is developed and deployed on FPGA. During the period of training, the network is optimized through pruning, fusion, and the integration of different attention mechanisms to achieve higher accuracy. The weight parameters are modularly divided based on hardware deployment requirements. In terms of architecture design, a data transmission architecture is proposed. A ping-pong buffer is utilized, combined with multi-DMA transmission, and a block strategy for off-chip memory access is included. Multiple accelerated IP cores are designed to align with our transmission architecture,

TABLE VI
COMPARISON OF YOLO ACCELERATION-RELATED WORK

| Model | SimYOLO [33] | YOLOv2 [34] | Tiny-YOLOv2 [35] | TinyYOLOv3 [16] | **This work** |
|---|---|---|---|---|---|
| Platform | GTX titan X | Intel corei7-6700k | CycloneV | Zedboard | **PYNQ-Z2** |
| Frequency | 1GHz | 4000MHz | 117 MHz | 100MHz | **100MHz** |
| Precision | float32 | float32 | fixed16 | fixed16 | **fixed16** |
| Performance(GOPS) | 1512 | 0.377 | 19.45 | 10.5 | **26** |
| Power(W) | 170 | 65 | – | 3.36 | **2.4** |
| Power efficiency(GOPS per W) | 8.89 | 0.0058 | – | 0.32 | **11** |

TABLE VII
COMPARISON OF OTHER ACCELERATORS

| Model | Ref[36] | Ref[37] | Ref[38] | **This work** |
|---|---|---|---|---|
| Platform | PYNQ-Z2 | Aritix7 | XC7045 | **PYNQ-Z2** |
| Frequency | 100HMz | 100Mhz | 150MHz | **100MHz** |
| Precision | fixed16 | fixed16 | fixed8 | **fixed16** |
| Power(W) | 1.652 | 7.53 | 10 | **2.4** |
| Performance (GOPS) | 3.422 | 22 | 38.4 | **26** |
| Power efficiency (GOPS per W) | 2.07 | 2.92 | 3.84 | **11** |

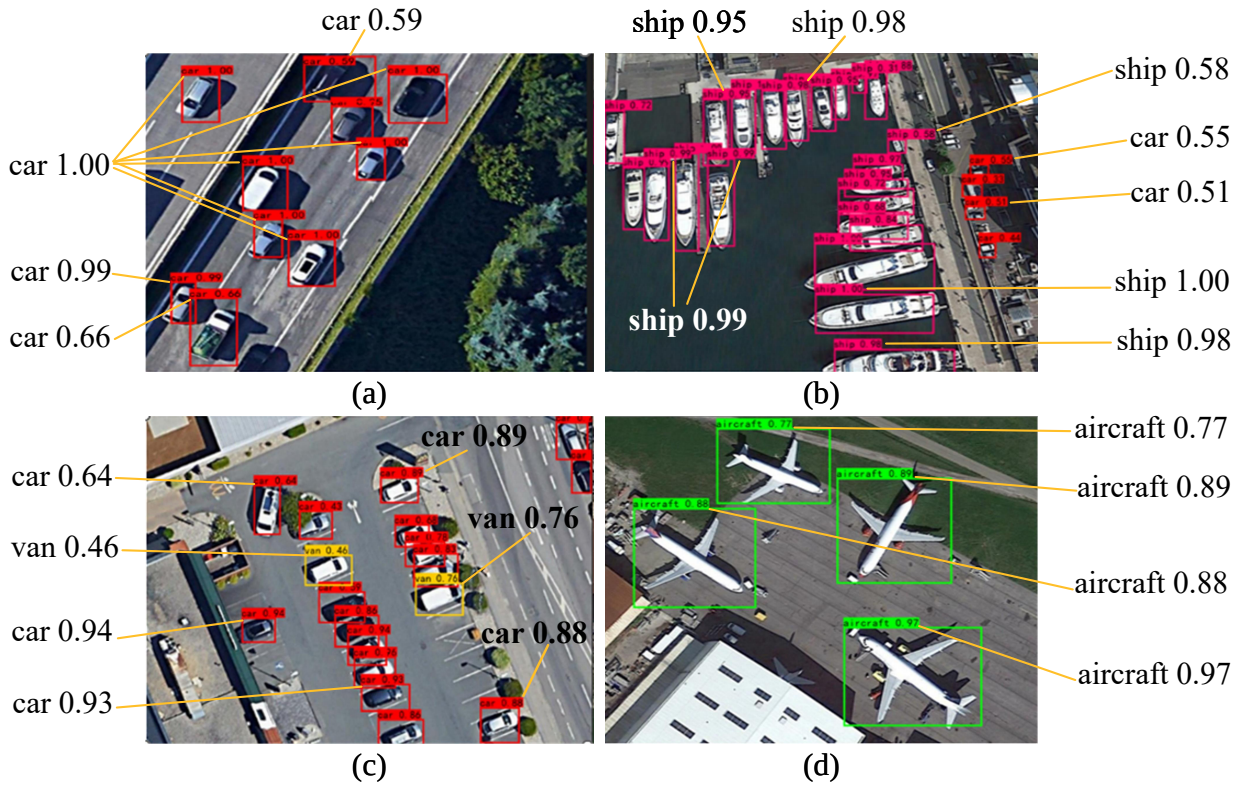


Fig. 16. Actual effect detection diagram

resulting in reduced data transmission delays. Regarding system verification, the SIMD dataset is used to evaluate our accelerator, and we integrated the intelligent transportation application scenarios. The results confirm that our designed accelerator maintains high accuracy in complex scenes. In conclusion, our proposed acceleration method reduces system power consumption, enhances energy efficiency, and achieves high identification accuracy, making it applicable for object detection in complex scenarios. Thus, it promises significance in hardware platform acceleration and object detection.

## REFERENCES

[1] S. Castillo, A. Bernal, and J. Rodríguez, "Object Detection in Digital Documents Based on Machine Learning Algorithms," *IAENG International Journal of Computer Science*, vol. 50, no. 2, pp. 688–699, 2023.

[2] D. Kaur, S. Uslu, K. J. Rittichier, and A. Durresi, "Trustworthy Artificial Intelligence: A Review," *ACM computing surveys (CSUR)*, vol. 55, no. 2, pp. 1–38, 2022.

[3] Y.-K. Wang, H.-Y. Syu, Y.-H. Chen, C.-S. Chung, Y. S. Tseng, S.-Y. Ho, C.-W. Huang, I.-C. Wu, and H.-C. Wang, "Endoscopic Images by A Single-Shot Multibox Detector for The Identification of Early Cancerous Lesions in The Esophagus: A Pilot Study," *Cancers*, vol. 13, no. 2, p. 321, 2021.

[4] Y. Chen, H. Wang, W. Li, C. Sakaridis, D. Dai, and L. Van Gool,

"Scale-Aware Domain Adaptive Faster R-CNN," *International Journal of Computer Vision*, vol. 129, no. 7, pp. 2223–2243, 2021.

[5] R. Deepa, E. Tamilselvan, E. Abrar, and S. Sampath, "Comparison of YOLO, SSD, Faster RCNN for Real Time Tennis Ball Tracking for Action Decision Networks," in *2019 International conference on advances in computing and communication engineering (ICACCE)*. IEEE, 2019, pp. 1–4.

[6] T. Fukagai, K. Maeda, S. Tanabe, K. Shirahata, Y. Tomita, A. Ike, and A. Nakagawa, "Speed-Up of Object Detection Neural Network with GPU," in *2018 25th IEEE International conference on image processing (ICIP)*. IEEE, 2018, pp. 301–305.

[7] S. Y. Neyaz, I. Saxena, N. Alam, and S. A. Rahman, "FPGA and ASIC Implementation and Comparison of Multipliers," in *2020 International Symposium on Devices, Circuits and Systems (ISDCS)*. IEEE, 2020, pp. 1–4.

[8] P. Hobden, S. Srivastava, and E. Nurellari, "FPGA-based CNN for Real-time UAV Tracking and Detection," *Frontiers in Space Technologies*, vol. 3, p. 878010, 2022.

[9] R. T. Kumar, S. Abinaya, D. Prakash, N. Janaki, S. Sivarajan, and P. Mani, "FPGA Interfaced IoT System for Smart Medical Robot Monitoring System," in *2024 2nd International Conference on Computer, Communication and Control (IC4)*. IEEE, 2024, pp. 1–6.

[10] A. A. Yazdeen, S. R. Zeebaree, M. M. Sadeeq, S. F. Kak, O. M. Ahmed, and R. R. Zebari, "FPGA Implementations for Data Encryption and Decryption Via Concurrent and Parallel Computation: A Review," *Qubahan Academic Journal*, vol. 1, no. 2, pp. 8–16, 2021.

[11] R. Dong, D. Xu, J. Zhao, L. Jiao, and J. An, "Sig-NMS-based Faster R-CNN Combining Transfer Learning for Small Target Detection in VHR Optical Remote Sensing Imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 11, pp. 8534–8545, 2019.

[12] Z. Jiang, L. Zhao, S. Li, and Y. Jia, "Real-Time Object Detection Method for Embedded Devices," in *computer vision and pattern recognition*, vol. 3, 2020, pp. 1–11.

[13] P. Hobden, S. Srivastava, and E. Nurellari, "FPGA-based CNN for Real-time UAV Tracking and Detection," *Frontiers in Space Technologies*, vol. 3, p. 878010, 2022.

[14] K. M. Ali, I. Alouani, A. A. El Cadi, H. Ouarnoughi, and S. Niar, "Cross-layer CNN Approximations for Hardware Implementation," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th International Symposium, ARC 2020, Toledo, Spain, April 1–3, 2020, Proceedings 16*. Springer, 2020, pp. 151–165.

[15] D. T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019.

[16] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized Neural Network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[17] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[18] T. B. Preußer, G. Gambardella, N. Fraser, and M. Blott, "Inference of Quantized Neural Networks on Heterogeneous All-Programmable Devices," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 833–838.

[19] Z. Yu and C.-S. Bouganis, "A Parameterisable FPGA-Tailored Architecture for YOLOv3-Tiny," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 16th International Symposium, ARC 2020, Toledo, Spain, April 1–3, 2020, Proceedings 16*. Springer, 2020, pp. 330–344.

[20] Y. Li, X. Dong, and W. Wang, "Additive Powers-of-Two Quantization: An Efficient Non-uniform Discretization for Neural Networks," in *International Conference on Learning Representations*, 2020.

[21] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to Quantize Deep Networks by Optimizing Quantization Intervals with Task Loss," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4350–4359.

[22] J. Zhang, L. Cheng, C. Li, Y. Li, G. He, N. Xu, and Y. Lian, "A Low-Latency FPGA Implementation for Real-Time Object Detection," in *2021 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[23] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An Automated Framework for Mapping Deep Neural Networks Onto FPGAs with RTL-HLS Hybrid Templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 152–159.

[24] S. Zhang, J. Cao, Q. Zhang, Q. Zhang, Y. Zhang, and Y. Wang, "An FPGA-based Reconfigurable CNN Accelerator for YOLO," in *2020 IEEE 3rd international conference on electronics technology (ICET)*. IEEE, 2020, pp. 74–78.

[25] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 45–54.

[26] D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler Transformations for High-Performance Computing," *ACM Computing Surveys (CSUR)*, vol. 26, no. 4, pp. 345–420, 1994.

[27] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.

[28] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing The Convolution Operation to Accelerate Deep Neural Networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

[29] J. Wu, B. Zheng, Y. Nie, and Z. Chai, "FPGA Accelerator for 3DES Algorithm Based on OpenCL," *Comput. Eng*, vol. 47, pp. 147–155, 2021.

[30] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA-based Neural Network Inference Accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[31] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays*, 2016, pp. 26–35.

[32] D. Wan, R. Lu, S. Shen, T. Xu, X. Lang, and Z. Ren, "Mixed Local Channel Attention for Object Detection," *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106442, 2023.

[33] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[34] F. Bi and J. Yang, "Target Detection System Design and FPGA Implementation Based on YOLO v2 Algorithm," in *2019 3rd International Conference on Imaging, Signal Processing and Communication (ICISPC)*. IEEE, 2019, pp. 10–14.

[35] J. W. Yap, Z. bin Mohd Yussof, S. I. bin Salim, and K. C. Lim, "Fixed Point Implementation of Tiny-YOLO-v2 Using Opencl on FPGA," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, 2018.

[36] Y. Shi, T. Gan, and S. Jiang, "Design of Parallel Acceleration Method of Convolutional Neural Network Based on FPGA," in *2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. IEEE, 2020, pp. 133–137.

[37] Q. Zhang, J. Cao, Y. Zhang, S. Zhang, Q. Zhang, and D. Yu, "FPGA Implementation of Quantized Convolutional Neural Networks," in *2019 IEEE 19th International Conference on Communication Technology (ICCT)*. IEEE, 2019, pp. 1605–1610.

[38] B. Khabbazan and S. Mirzakuchaki, "Design and Implementation of A Low-power, Embedded CNN Accelerator on A Low-end FPGA," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 647–650.