FPGA-based Object Detection Acceleration Architecture Design

FPGA-based Object Detection Acceleration Architecture Design

Wenhao Li[1,a], Huaixiang HU[1]

[1]North China Institute of Computing Technology

[a]lwhtjdx@163.com

Abstract. To solve the problems of insufficient acceleration capability and high power consumption of the existing embedded terminal target detection, a multi-strategy optimization hardware accelerator based on the YOLOv2 model is designed by taking advantage of the high concurrency characteristics of the CPU+FPGA structure. The accelerator improves its parallelism and real-time performance while reducing power consumption and resource consumption through floating-point quantization, adder optimization, and various HLS optimization strategies. Tested on Xilinx's PYNQ-Z2 platform, the overall computing power of the accelerator reaches 27.1GOP/s, the average detection accuracy is 80.6%, and the overall power consumption is 2.609W. Compared with CPU and GPU, the obtained accuracy only loses 2 %, but the power consumption is greatly reduced. It has strong practical significance in edge target detection that requires high real-time performance and low power consumption.

1. Introduction

Among the many technical fields of computer vision, object detection is a very basic task, and it has important applications in intelligent security, autonomous driving, industrial inspection, aerospace, and other fields [1]. The generalization ability of traditional computer vision is weak, and deep learning technology has gradually developed and become mainstream because of its better performance. Most of the target detection models cannot perform edge real-time target detection because of their huge computing and storage capacity, which makes it difficult for embedded terminals to meet their hardware requirements. As a network with a simple structure and outstanding performance and real-time performance, the YOLOv2 network is an excellent choice to land it on edge target detection.

A separate CPU is more suitable for processing serial control flow, but not suitable for large-scale parallel computing; a separate GPU is more suitable for processing large-scale parallel computing, but it is stretched in terms of processing control flow. Multi-core ARM is suitable for video acquisition and data preprocessing, and various acceleration operators in FPGA are suitable for feature extraction and feature comparison. It has low power consumption at the same time as high parallel computing, and its hardware-programmable feature makes it less difficult to develop. The development cycle is short.

Aiming at the problems of insufficient real-time performance, high power consumption, and insufficient storage and computing resources of current network models deployed in embedded terminals, this paper proposes a multi-strategy optimized hardware accelerator. By utilizing the logic processing capability of the CPU and the high parallel computing resources of the FPGA, it can simplify the model structure and improve the parallelism without reducing the accuracy of the model itself, to realize a high-performance, high-real-time, and low-power accelerator. It also has advantages in accuracy and resource consumption in comparison with other platforms and other models.

2. YOLOv2 Model introduction

YOLO is a new framework proposed to improve the real-time performance of target detection after RCNN and fast-RCNN[2]. YOLO uses the entire image as the input of the network, and directly returns the position and category of the prior frame in the output layer, as shown in Figure 1.

The YOLO model regards target detection as a regression problem. The entire image is input, the image features are extracted, and all classes and bounding boxes are predicted by the image regression method to obtain the final probability and bounding box coordinates. This means that global information can be extracted in end-to-end training detection. For the overall training, the detection speed is greatly improved.

The YOLOv2 network consists of convolutional layers, pooling layers, routing layers, and reordering layers. The convolutional layer is used to extract different features of the input image; the pooling layer is used to reduce the feature dimension, compress the number of data and parameters, reduce overfitting, and improve the fault tolerance of the model; the routing layer is used before the current layer is drawn The feature layer obtained by convolution fuses multi-layer features; the reordering layer is used to pass all the information to the subsequent layers, and there is no loss of information due to operations such as downsampling, so more fine-grained features are obtained. The training is accelerated to converge by batch normalization at training time, and all nonlinear functions are fitted by the activation function.

YOLOv2 adopts the Darknet19 network structure, 19 layers of convolution, and 5 layers of pooling. The input image is increased from 224×224 of Darknet19 to 416×416. The last layer of Darknet19 convolution and average pooling are removed and replaced with two layers of convolution. The number of layers is less than that of YOLOv1, and there is no fully connected layer, so the calculation amount is less, and the model runs faster. The use of convolution instead of full connection removes the limitation of input size, and multi-scale training makes the model adaptable to different scales. Image detection is more robust; each unit uses 5 anchor boxes for prediction, which is more effective for crowded and small object detection.

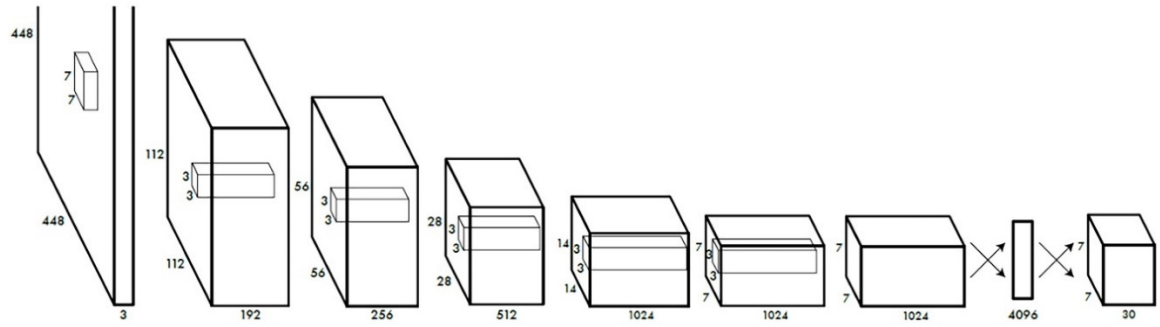

Figure 1. YOLOv2 network structure.

3. Hardware acceleration optimization strategy

3.1. floating point quantization

During model training, the network parameters and intermediate results of the YOLO network model are single-precision floating-point numbers. The YOLO model accelerated requires a lot of logic resources, making it extremely difficult to deploy online in a high-throughput production environment. Due to the integration of a large number of hardware fixed-point multipliers with variable precision on the FPGA, the multipliers have also been deeply optimized [3]. So on the FPGA, the fixed-point computing resources and power consumption costs are much smaller than floating-point arithmetic. The fixed point of the YOLO model can reduce the resource consumption of the FPGA, and if the same resources are used, the performance of the accelerator can be greatly improved.

Quantization and inverse quantization of floating-point numbers:

$$r = S(q - Z) \tag{1}$$

$$q = round(\frac{r}{S} + Z) \tag{2}$$

Where r represents a floating-point real number, q represents a quantized fixed-point integer, S is a scale, which represents the proportional relationship between a real number and an integer, and Z is a zero point, which represents the quantized integer corresponding to 0 in the real number, its calculation method for:

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} \tag{3}$$

$$Z = round(q_{max} - \frac{r_{max}}{S}) \tag{4}$$

Fixed-pointing of the YOLO model, mainly for the fixed-pointing of convolutional layer parameters and activation maps. The parameters obtained after convolution can only be fixed-pointed once and can be stored after quantization. The activation map result is determined by the convolution layer parameters and the input image, and its dynamic range is much larger than that of the convolution parameters, so it needs to be re-quantized every time, as shown in Figure 2.

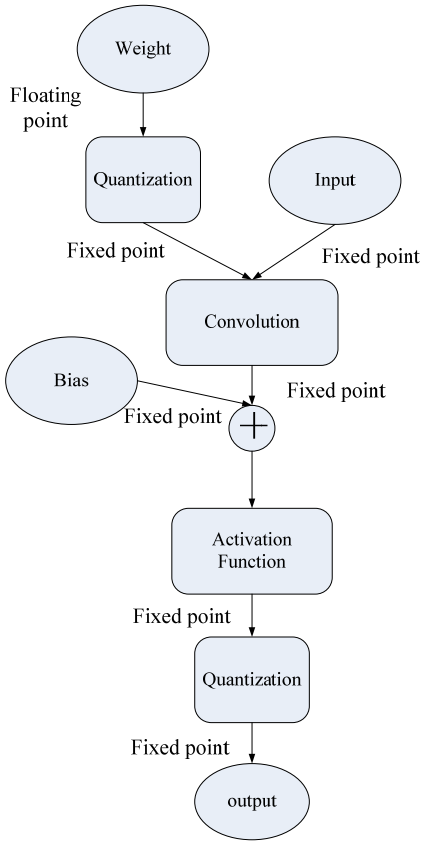

Figure 2. Floating point quantization process.

3.2. Optimization based on HLS

The optimization principle of HLS is not to affect the actual function of the algorithm and to parallelize the algorithm as much as possible within the resource range permitted by the FPGA chip to reduce time delay and improve throughput.

3.2.1. Loop optimization.

In most algorithms, the loop is an inevitable part, and the way to realize the loop will affect the efficiency and resource occupation of the FPGA chip. Therefore, the cycle needs to be analyzed. Common loop optimization methods include Pipeline, Unroll, and Dataflow.

Pipeline optimization can improve latency and interval. When there is no pipeline, the entire operation is executed in chronological order. With the pipeline, the hardware circuit is divided into n equal stages, and the next operation can start without waiting for the previous operation to finish. Different stages can be executed consecutively and in parallel, as shown in Figure 3. Based on the above description, the pipeline structure can improve the delay and increase the system throughput without increasing the hardware resource overhead too much, and makes a good compromise in the balance between resources and performance.

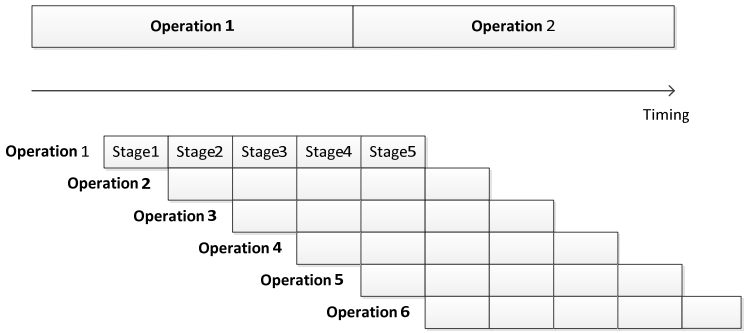

Figure 3. Pipeline structure.

Unroll means loop unfolding. The circuit is folded (rolled) by default in a loop. It can be understood that the same set of circuits is used for each loop [5], and each loop is time-division multiplexed. If Unroll expands the circuit, the loop circuit will be copied several times, and the specific number can be set by parameters. These duplicated circuits can execute different data simultaneously. The system throughput is greatly improved, but resource consumption will also increase exponentially.

Dataflow implements task-level pipelines outside of loops so that certain loops and functional functions can still overlap in some conflict (such as data-related) situations, increasing the concurrency of RTL designs, and thereby improving overall throughput. As shown in Picture 4.

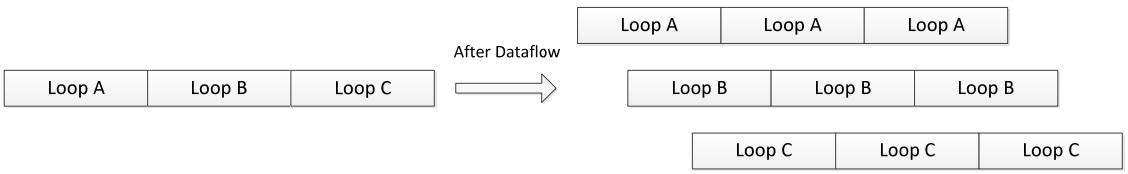

Figure 4. Dataflow optimization.

3.2.2. Array optimization.

In HLS, arrays are usually synthesized as memory (RAM, ROM, or FIFO). Like loops, arrays can add instructional code to the code and perform corresponding RTL optimizations. The interface to an array is often a performance bottleneck because when an array is implemented in memory, the number of memory ports limits the rate at which data can be obtained. At this point, the array can be divided and the number of memory ports can be increased to increase the bandwidth.

There are three ways to divide the array, which are block, cyclic and complete. Block division is to divide the entire array of elements into a certain number of blocks, and fill each block in sequence. Cyclic segmentation is to cyclically allocate elements for the specified number of blocks until all

allocations are complete. Complete segmentation is to allocate a register for each element separately for storage.

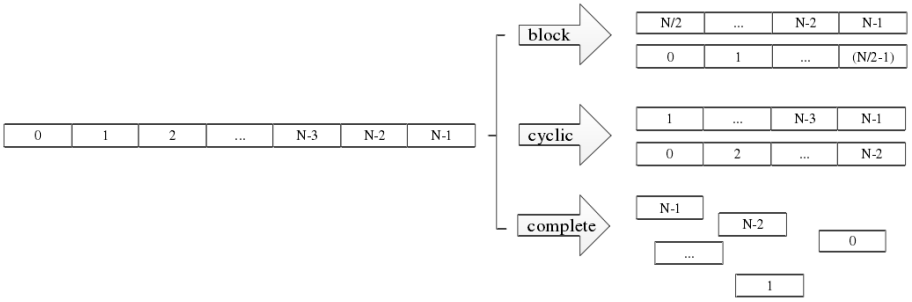

Figure 5. Multiple array division methods.

As shown in Figure 5, an array of length N is divided into blocks. When using the block division method, specify the block parameter factor=2, and HLS will divide the entire array into two consecutive arrays of length N/2. When using the cyclic division method, also specify the block parameter factor=2, and HLS will divide the entire array into two alternately growing arrays of length N/2. When the complete splitting method is used, HLS splits the entire array into individual elements by default, and each element is independently stored in a register. Therefore, the block and cyclic segmentation methods can double the number of ports of the module, while the complete segmentation method can increase the number of ports of the module by N times, and the data throughput is greatly improved, but the resource consumption of on-chip registers is also greatly increased.

4. system design

4.1. The specific design of each module

YOLOv2 consists of 24 layers, involving convolutional layers, max-pooling layers, and finally a detection layer. Among them, the convolution layer plays the role of feature extraction, and the pooling layer is used to sample and reduce the size of the feature map.

4.1.1. Convolution Module Design.

The processing performed by the convolutional layer is the convolution operation. The convolution operation is equivalent to the "filter operation" in image processing. The input convolution image is extracted according to the corresponding convolution kernel, and the pseudo-code is as follows:

Convolution module operation

```
for(int n = 0 ; n < N ; n ++)
    for(int i = 0 ; i < Hout ; i ++)
        for(int j = 0 ; j < Wout ; j ++)
            for(int ii =0 ;ii<Ky ; ii ++)
                for(int jj = 0 ; jj < Kx ; jj ++)
                    for(int m = 0 ; m < M ; m ++){
                        tp = cin_feature[h][w][m]*w[ii][jj][m][n];
                        m += tp;}
            sum = sum + bias;
```

Where (Hout, Wout), CHout, CHin, (Kx, Ky) represent the output feature map, the number of output feature maps, the number of input feature maps, and the size of the convolution kernel, respectively. The sum represents the pixels in the output feature map.

In the hardware sequential logic, the input and output feature maps are expanded in two dimensions, Nin input buffers, and Nout output buffers can be set. And there will be Nin*Nout parallel multiplication units and Nout addition trees, Nin input features After passing through the parallel multiplication unit, the final output feature map is obtained through the addition tree and stored in the output buffer. The entire convolution computation is pipelined through a parallel approach.

4.1.2. Pooling Module Design.

In a convolutional neural network, a pooling layer is usually added between adjacent convolutional layers. The pooling layer can effectively reduce the size of the parameter matrix, thereby reducing the number of parameters in the final connection layer. Therefore, a pooling layer added can speed up the calculation and prevent overfitting. YOLOv2 uses a maximum pooling layer, and the pseudocode of the maximum pooling layer is as follows:

```
Pooling Module Operations
for(int i = 0 ; i < Hout ; i ++)
for(int j = 0 ; j < Wout ; j ++)
   for(int n = 0 ; n < N ; n++)
      for(int ii =0 ;ii<Ky ; ii ++)
         for(int jj = 0 ; jj < Kx ; jj ++)
               Sum = (ii == 0 && jj == 0) ? MIN : MAX(sum,tp) ;
```

The maximum or minimum constant value is returned by the Max function and the Min function. The other operation logic is similar to convolution, except that the multiplication and addition operation of the last layer is converted into a comparison operation, which is much smaller than the number of convolution operations in the overall proportion, just to reduce the number of parameters obtained by the convolution layer.

4.2. Overall system design

The YOLO target detection algorithm is mainly divided into three steps: preprocessing, network derivation, and postprocessing: 1) Preprocessing: for the input RGB image of any resolution, the pixel value of each channel pixel is normalized to [0,1] interval, and scale the image size to 416×416 according to the aspect ratio of the original image. 2) Network derivation: Input the normalized 416×416×3 image into the YOLOv2 network for forwarding derivation and get a 13×13×5×25 output tensor. 3) Post-processing: According to the format of the output tensor, the center point coordinates and length and width of each frame are obtained, and NMS processing is performed on all 13×13×5 frames to obtain the candidate frame of the target object.

The network structure and algorithm steps of YOLOv2 analyzed can be seen that the maximum amount of computation is convolution and pooling. The convolution layer uses convolution kernels of different sizes to perform feature extraction on the input feature map, and a pooling layer is set after the convolution layer, to simplify the complexity of the model, and reduce the feature map by downsampling the feature map. Therefore, when designing based on a partial hardware-based design method, it is only necessary to use HLS to design the convolution IP core and the pooling IP core. From the perspective of system function and composition, the YOLOv2 hardware acceleration system shown in Figure 6 can be divided into four parts: storage subsystem, computing subsystem, control path, and data path. The storage subsystem includes the Cache in the CPU and the on-board DDR3 memory of the PYNQ-Z2; the computing subsystem includes the Cortex-A9 CPU on the PS side and the convolution IP core and the pooling IP core in the FPGA on the PL side; the control path and data path refer to the paths that the control signals and data signals in the acceleration system travel through when they are transmitted.

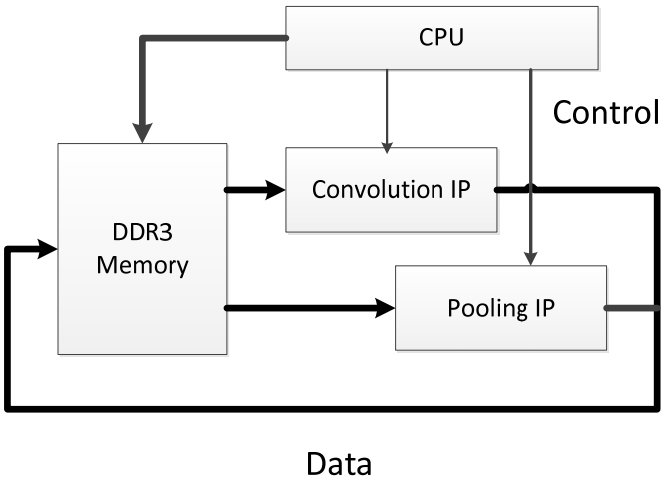

Figure 6. Hardware Accelerator Architecture.

Generally, the forward derivation of the network using the YOLO hardware accelerator needs to go through three stages system initialization, data processing, and result recovery. In the system initialization stage, the CPU on the PS side first divides the DDR3 storage space into several partitions, which are used to store YOLO network parameters, input image data, intermediate calculation results, and network derivation results; then YOLO network parameters are written through the AXI bus into the designated partition of the DDR3 memory. In the data processing stage, the CPU on the PS side writes the image to be recognized into the DDR3 memory, and sends out corresponding control signals according to the forward derivation process of YOLO to control the convolution IP core and the pooling IP core on the PL side to continuously read DDR3 Network parameters and input image data in the memory, and perform corresponding calculations; the intermediate results generated during the calculation process will also be stored in DDR3. After the CNN forward derivation is completed, the derivation result will be stored in the designated partition of DDR3, and the result recovery stage will be entered at this time. The CPU on the PS side reads the derivation result in DDR3 through the AXI bus, parses the derivation result, and finally outputs and displays the obtained image recognition result.

Based on the above system framework, HLS is used to generate convolution and pooling IP, and the entire YOLO target recognition hardware acceleration system is built through vivado. As shown in Figure 7, ZYNQ's ARM processor links the YOLO_FPGA IP core through the AXI interconnector, completes data control and reading and writing through the AIX interface, and finally completes the preceding derivation of image data. The final recognition result is shown in Figure 8.

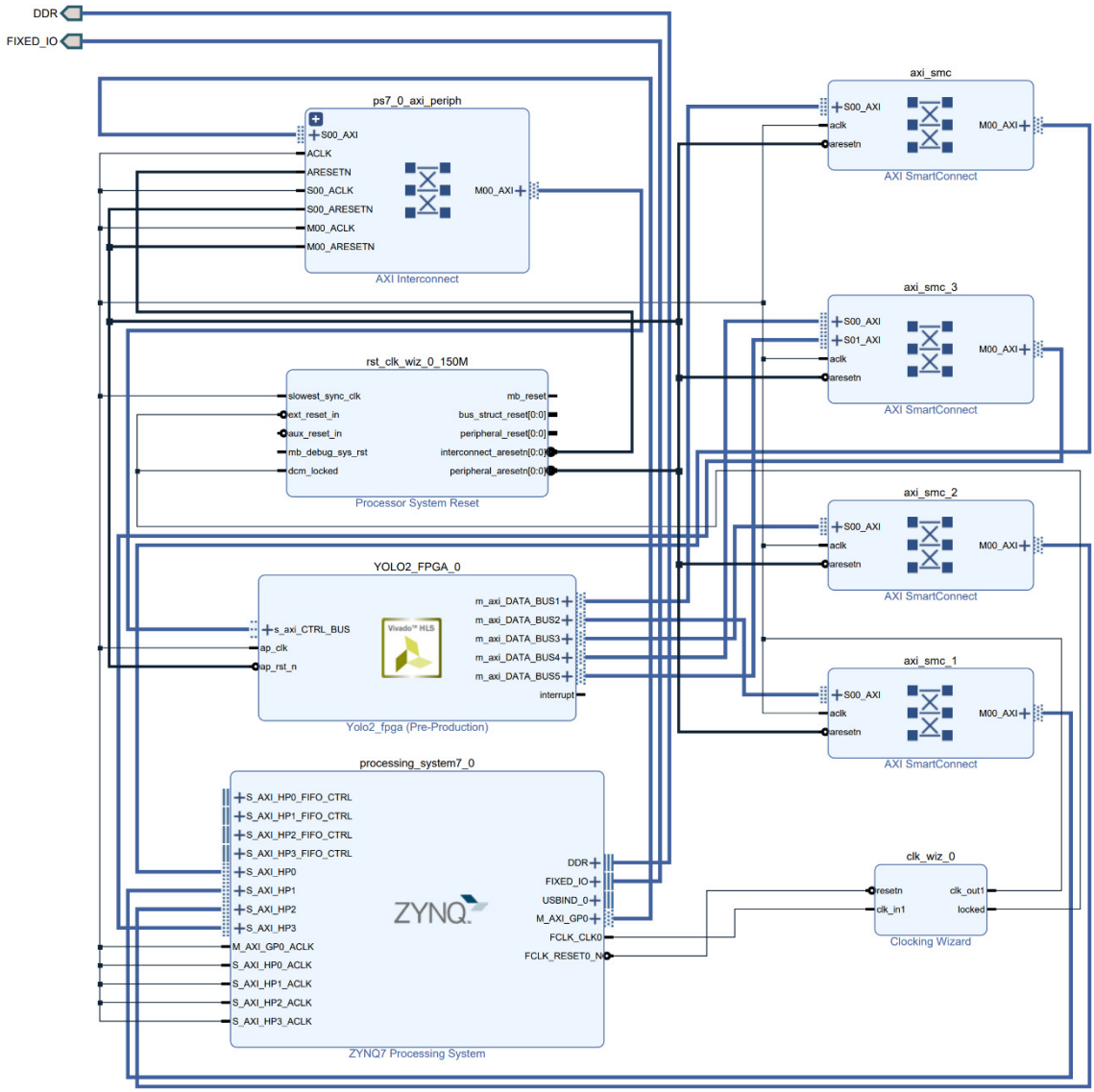

Figure 7. YOLOv2 Model Hardware Accelerator.

Figure 8. Recognition result comparison chart.

5. Experimental evaluation

5.1. Resource consumption

In the experiment, the acceleration platform selected is PYNQZ2 of Xilinx Company, its main chip is ZYNQXC7Z0201CLG400, and its processor is ARM dual-core Cortex-A9. After the overall architecture is implemented in vivado2018.3, the resource consumption of each module is shown in Figure 9. The utilization rate of DSP is the highest, reaching 67.7% because a large number of multiplication operations in the convolution calculation are completed by DSP. BRAM is used to store intermediate parameters and convolution results, and 175 are used. After adder optimization, the number of LUTs and FFs used is greatly reduced. LUTs mainly implement operations such as addition and subtraction and logical shifts, using 35373, while FFs are mainly used as registers, using 32555.

Resource	Utilization	Available	Utilization %
LUT	35373	53200	66.49
LUTRAM	6158	17400	35.39
FF	32555	106400	30.60
BRAM	87.50	140	62.50
DSP	149	220	67.73
BUFG	3	32	9.38
MMCM	1	4	25.00

Figure 9. Accelerator resource consumption.

5.2. Performance Power Assessment

Randomly select photos from the COCO data set for input. After the system experiment, the overall computing power of the accelerator reaches 27.1GOP/s, the average detection accuracy is 80.6%, and the overall power consumption is 2.609W, of which the dynamic power consumption is 2.427W, and the static power consumption is 0.183W. As shown in Figure 10.

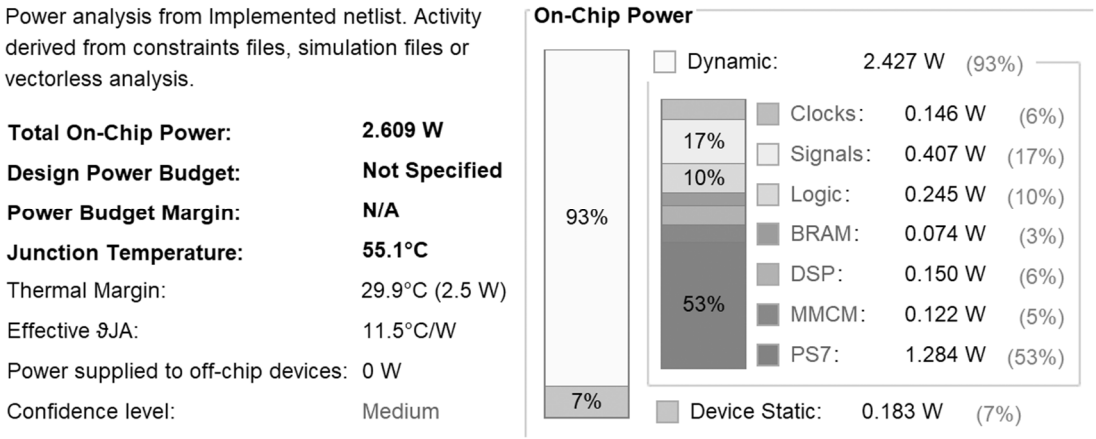

Figure 10. Accelerator power consumption.

After the experimental comparison of different processor platforms, the CPU and GPU were tested with darknet19, developed through C language, and all operations were floating-point numbers. FPGA used the Zynq platform to develop IP through HLS and quantized floating-point into fixed-point. Compared with the CPU and GPU, the loss is only 2%, but the power consumption is greatly reduced, as shown in Table 1.

Table 1. Performance comparison of different hardware platforms.

	CPU	GPU	FPGA
Platform	Intel core i7	GeForce 1080ti	Zynq-7000
Development language	C	C	HLS
mPA	82.60%	82.40%	80.60%
Data Precision	Float	Float	Int8
Power /W	122	249	2.6

Finally, this accelerator is compared with the FPGA-based YOLO neural network accelerator released by other researchers, as shown in Table 2. It can be seen that after floating-point optimization, adder optimization, and various parallel optimizations, the power consumption is almost the lowest, and the number of DSPs used is greatly reduced, leaving enough room for subsequent post-processing, and not only achieving 80.6 in performance of the highest accuracy, it also has a relatively strong computing power of 27.1GOP/s and a good data throughput.

Table 2. Comparison of YOLO accelerators in different kinds of literature.

	YOLOv1[7]	Tiny YOLOv2[9]	Lightweight YOLOv2[10]	This Work
Platform	ZC706	Virtex-7 VC707	Zynq Ultrascale+	Pynq-z2
DSP	800	168	377	149
mPA	N/A	51.40%	67.60%	80.60%
Power/W	2.17	8.7	N/A	2.61
Performance/GOPs	18.82	N/A	N/A	27.1

6. Conclusion

In this paper, the YOLO-v2 network is used as the model and the Pynq-z2 is used as the development platform to design and test the hardware network acceleration circuit serving the model. At the same time, the method of floating point quantization and adder optimization is used to reduce the hardware resource consumption of the circuit without reducing the network detection accuracy, so that it can be deployed in the terminal for edge detection. A variety of parallel optimization and array optimization in HLS is used to improve the parallelism of the algorithm while sacrificing a certain area, the circuit delay and performance are improved, the overall computing power is increased, and the circuit power consumption is reduced. Finally, 80.6% mPA, 27.1GOP/s overall computing power, and 2.6W low power consumption can be obtained. In comparison with different platforms, a more suitable balance point can be found in performance, accuracy, and power consumption, which satisfies the need for faster and more accurate target detection under the premise of having fewer resources, and realizes the YOLO-v2 terminal.

References

[1] Girshick R, Donahue J, Darrell T, et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation [C]// IEEE Computer Society. IEEE Computer Society, 2013, 580–587.

[2] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection [C]// IEEE Conference on Computer Vision and Pattern Recognition. Piscataway, 2016: 779-788.

[3] CHAN KIM, HYUN MI KIM, CHUN-GI LYUH, et al. Implementation of Yolo-v2 Image Recognition and Other Testbenches for a CNN Accelerator [C]// 2019 IEEE 9th International Conference on Consumer Electronics: ICCE-Berlin 2019: 242-247.

[4] Chen T, Du Z, Sun N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning [J]. Acm Sigplan Notices, 2014, 49(4): 269-284.

[5] FANGHONG BI, JUN YANG. Target Detection System Design and FPGA Implementation Based on YOLO v2 Algorithm [C]// 2019 3rd International Conference on Imaging, Signal Processing, and Communication: ICISPC 2019:11-15.
[6] YU, YUNXUAN, WU, CHEN, ZHAO, TIANDONG, et al. OPU: An FPGA-Based Overlay Processor for Convolutional Neural Networks [J]. IEEE transactions on very large scale integration (VLSI) systems, 2020, 28(1): 35-47.
[7] XI LIN, YANNAN MO, TAO SU. Failure Characteristics of FPGA-Based Convolutional Neural Networks under RF Interference [C]// 2021 IEEE 4th International Conference on Electronics Technology: IEEE 4th International Conference on Electronics Technology (ICET), 2021: 364-368.
[8] Zhao R, Niu X, Wu Y, et al. Optimizing CNN-Based Object Detection Algorithms on Embedded FPGA Platforms [C]// International Symposium on Applied Reconfigurable Computing. Delft, 2017: 32-42.
[9] Nguyen D T, Nguyen T N, Kim H, et al. A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection [J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019, 27(8): 1-13.
[10] Nakahara H, Yonekawa H, Fujii T, et al. A Lightweight YOLOv2: A Binarized CNN with A Parallel Support Vector Regression for an FPGA [C]// the 2018 ACM/SIGDA International Symposium. USA, 2018: 238-243.