Journal of Computers Vol. 32 No. 5, 2021, pp. 115-127
doi:10.53106/199115992021103205009

Low-power Iris Recognition System Implementation on FPGA with Approximate Multiplier

Meng-ru Lin[1,2], Shi-zhen Huang[1*], Fu-shan Li[1], Rui-qi Chen[2], Shi-di Tang[3]

[1] College of Physics and Information Engineering, Fuzhou University, Fuzhou 350116, China
n191127957@fzu.edu.cn, fushanli@hotmail.com, hs501@fzu.edu.cn

[2] VeriMake Innovation Lab, Nanjing Renmian Integrated Circuit Co., Ltd., Nanjing 210000, China
rickychen@verimake.com

[3] School of Geographic and Biological Information, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
1020173019@njupt.edu.cn

Received 30 January 2021; Revised 20 April 2021; Accepted 23 April 2021

Abstract. Covid-19 has been threatening human life and is now the most serious public health issue in the world. During this pandemic, wearing masks is one of the most effective ways to inhibit virus transmission. However, existing ubiquitous identity recognition requires people to remove their masks to complete facial recognition, which is highly risky. Iris recognition, as a safer applicable identification method, has its fatal weakness of not being able to achieve satisfactory real-time recognition on end devices. This paper presents an edge deployment of a low-power iris recognition system based on FPGA with approximate multipliers. We adopted a serial-parallel hybrid method for the preprocessing stage, trained the CNN model on PC and then deployed the architecture and parameters on FPGA. We further reduced power and resource consumption by designing approximate multipliers for the key calculation. Experimental results show that design achieves up 28% and 43% gain in terms of area and latency energy product, while incurring a negligible accuracy loss. The recognition speed increased by 40% compared with Raspberry Pi, 11 times better than Jetson Nano power latency production.

Keywords: iris recognition, FPGA, approximate multiplier, CNN

1 Introduction

During the Covid-19 period, the efficiencies of common identity (ID) information recognition methods are greatly reduced, such as facial recognition [1] and fingerprint recognition [2]. The reason is that with masks on people's face, key facial feature information cannot be properly obtained from the images, and the identification cannot be completed. On the other hand, fingerprint identification requires touching on the devices, which is quite risky during the epidemic as it greatly increases the risk of viral transmission [3]. However, temporary ID check points are needed in large quantities in hospitals, ticket barriers and other essential places. This poses huge challenges to the design of real-time edge devices for ID identification [4-5].

Iris recognition is another image-based way to achieve ID information recognition [6]. It is more suitable for ID identification during the pandemic, as it only needs to collect the image information of the iris, without the need of taking off masks or directly contacting with the detection devices. A complete iris recognition terminal system mainly has four steps, including acquisition of iris image, preprocessing of iris image, feature extraction and feature recognition. Moreover, its image resolution requirements are

* Corresponding Author

much higher compared with facial recognition. As a result, the device not only needs to be portable, low-power and low-cost, but also has to have strong computing power in order to realize real-time iris recognition [7].

In this paper, we implement an offline iris recognition system using field programmable gate array (FPGA) that achieves the above requirements in the greatest extend. We improved the iris recognition methods in existing researches, and expanded the data on the open-sourced iris database [8]. We also trained the recognition network and optimized the network model. Moreover, an approximate multiplier is designed to reduce resource consumption for the core operations in the identification process. Finally, it is deployed to FPGA terminal to realize local real-time iris recognition, with low power consumption and low resource consumption. Our main contributions are as follows:

1. The iris recognition architecture is optimized and deployed to FPGA to realize real-time iris recognition;
2. An approximate multiplier is designed based on power multiplication, which reduces the resource consumption with almost no change in identification accuracy;
3. Different embedded terminals are compared and analyzed with our designed FPGA iris recognition system;

The rest of this paper is arranged as follows: Section 2 introduces the system architecture and FPGA implementation of iris recognition proposed. Section 3 describes the introduced approximate multiplier in detail. Section 4 discusses the analysis of iris recognition accuracy and hardware resource consumption. Finally, the Section 5 summarizes the whole paper.

2 System Design

2.1 Overall Design of System

Fig. 1 shows the overall architecture of the system, including a high-definition camera with infrared light, a camera controller based on FPGA design, and an iris recognition accelerator. Firstly, the high-definition camera with infrared light is driven to collect the iris images, and they are cached to the SDRAM. Then, the preprocessing module in the accelerator makes the localization and normalization processing of iris images.

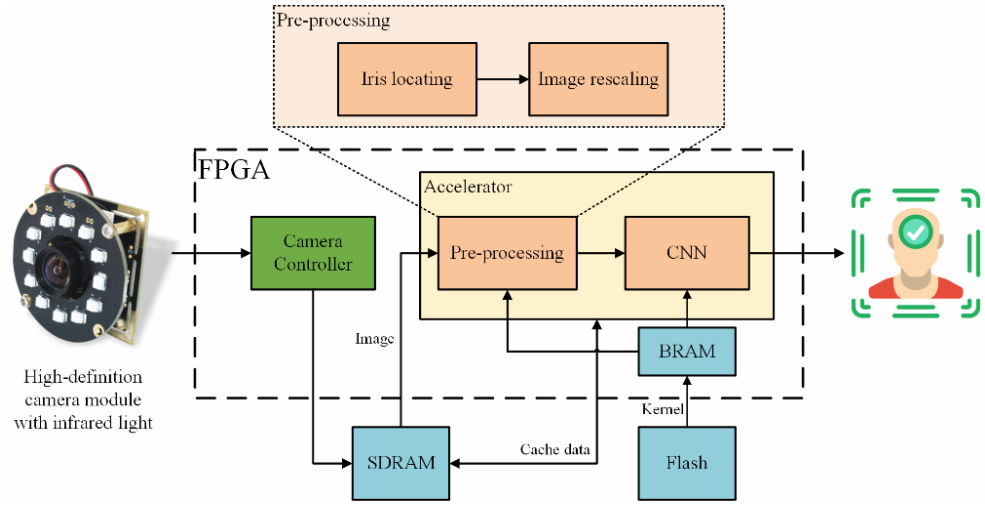


Fig. 1. The structure diagram of the proposed iris recognition system

Subsequently, the features are extracted and matched by a convolutional neural network (CNN) module. And finally, the results are output. It is worth noting that Kernel is involved in the operation of preprocessing module and the CNN module is stored by the external Flash. The data are controlled by a finite-state machine (FSM) and cached to BRAM for use. The image data in the operation are cached by SDRAM. The dataset is described in section 2.2. Furthermore, the design and implementation of the iris recognition accelerator which is the kernel of the system would be introduced in section 2.3.

2.2 Dataset Introduction

The system proposed in this paper is experimented and verified based on the dataset CASIA-iris-interval-v4.0 [8] and ND-IRIS-0405 of the near-infrared illumination. A single category of iris images are less in the dataset, and even there are two or three images in some categories. Such a dataset for the CNN training of iris could make the network framework enormous or underfitting, thus resulting in some problems such as low recognition accuracy, huge hardware resource consumption, etc. Here, the solution is to artificially expand the iris dataset. Namely, the dataset is expanded to 200 images at least in each category after adding the noise, rotation, and scaling [9-10].

2.3 Preprocessing

The first step of iris recognition refers to the preprocessing of the iris images, which is the most important foundation for the feature extraction and recognition. Here, the iris positioning and iris normalization are mainly implemented [11-12].

2.3.1 Iris Positioning

During the process of iris positioning, it is firstly needed to detect the internal and external boundaries of iris by the Canny edge filter, and then five steps, namely, Gradient computation based on Sobel operator, Gradient magnitude and orientation computation, Non-maximum suppression, and Adaptive threshold computation, Hysteresis thresholding, are completed by canny edge filter [13]. The structure of the Canny edge detector is shown in Fig. 2. The images are input by the Sobel operator in x and y directions, that is, the MAC module takes the multiply-accumulation.

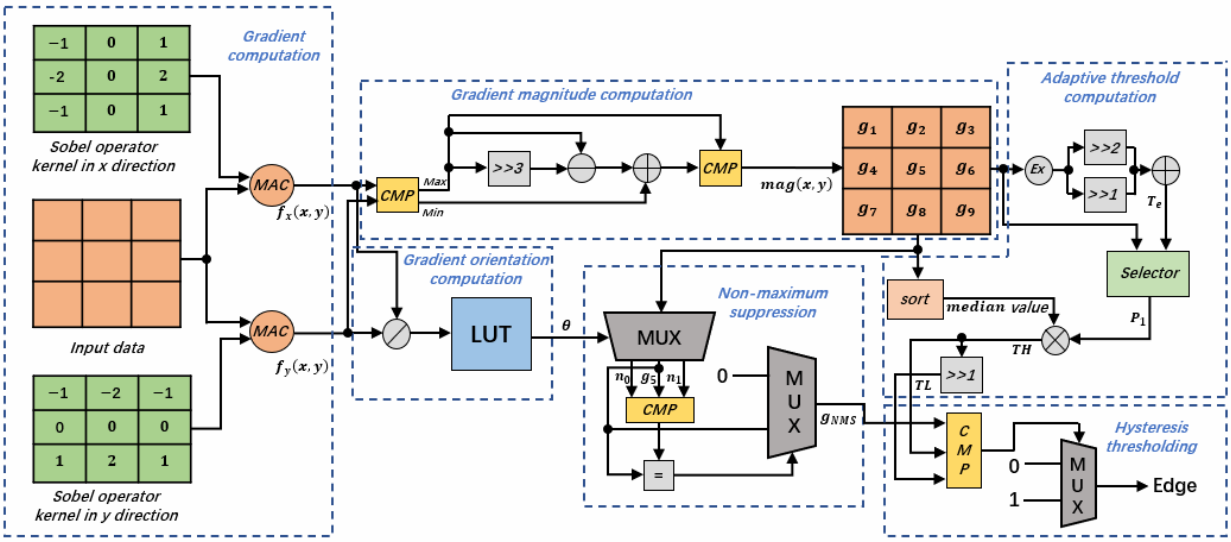


Fig. 2. Block diagram of the Canny edge detector

The operation is similar to that of the convolution layer mentioned in the following paragraphs, and there is no need to describe it in detail. Thus, the derivatives $f_x(x, y)$ and $f_y(x, y)$ in x and y directions are obtained. Then, in order to reduce the hardware consumption, the formulas (1)(2) are used to figure out the gradient amplitude $mag(x, y)$ and gradient direction $\theta(x, y) \in [0°, 180°]$. Moreover, a and b are the maximum and minimum values of $f_x(x, y)$ and $f_y(x, y)$, respectively. The angle from 0° to 180° is quartered, and the gradient direction $\theta(x, y)$ is approximately quantified in the four angle areas by lookup table (LUT). After center gradient amplitude g_5 and its adjacent gradient amplitudes $g_1 \sim g_4$ and $g_6 \sim g_9$ are obtained, it is necessary to get two adjacent gradient amplitudes (n_0 、 n_1) along the gradient direction $\theta(x, y)$. Subsequently, the three data are compared with each other. If the center gradient amplitude is the maximum, it could be reserved, otherwise, the center gradient amplitude is 0. Thus, the

non-maximum suppression operation is accomplished. Simultaneously, the adaptive threshold operation should be implemented. The mean and median of $g_1 \sim g_9$ are figured out with EX module and sort module. Thus, *Te*, high threshold, and low threshold are obtained by formulas (3), (4), and (5). Among them, P_1 is the percentage of strong edge pixels. The g_{NMS} less than TL is assigned to 0, but g_{NMS} more than TH is assigned to 1. Furthermore, the points, more than TL but less than TL, are determined by 8 connected regions. Namely, only connecting with the points more than TH can g_{NMS} be assigned to 1.

$$mag(x, y) = \max((0.875a + 0.5b), a) \tag{1}$$

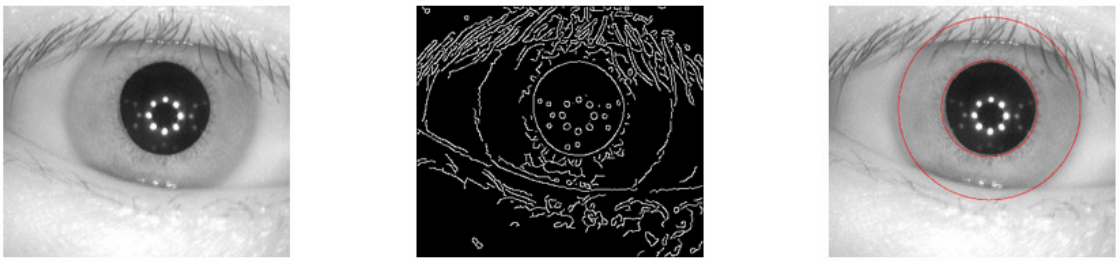
$$\tan\theta(x, y) = \frac{f_y(x, y)}{f_x(x, y)} \tag{2}$$

$$Te = 0.75 * Meanvalue \tag{3}$$

$$TH = P_1 * Meanvalue \tag{4}$$

$$TL = 0.5 * TH \tag{5}$$

After detecting the edges of internal and external boundaries, Circle Hough Transform is still used to mark the positions of internal and external boundaries of the iris [14]. That is, the radius and center of the inner and outer circles are obtained for the subsequent normalization operation. As can be seen in Fig. 3(b) is the iris image after the Canny edge filtering, in which the internal and external boundaries are clearly seen. Fig. 3(c) displays the iris after Circle Hough Transform. Hence, two approximate rings can determine the internal and external boundaries of the iris.



(a) raw picture (b)after Canny filtering (c) result

Fig. 3. Examples of the iris preprocessing

2.3.2 Iris Normalization

Here, it is needed to convert the Cartesian coordinates of the iris image to the polar coordinates, and then the positioned iris is unfolded into a rectangle with a specific unfolded way from the internal boundary of the iris to the external boundary. As can be seen in Fig. 4(a), the iris is incomplete for several lines at the bottom of unfolded rectangle due to the shade of eyelash and eyelid. Hence, the solution is to abandon the obscured part, and the effective and continuous iris is selected as the convolutional network image for input. The image size is determined as 200×40, as shown in Fig. 4(b).

(a) before normalization (b) after normalization

Fig. 4. Examples of the iris normalization

2.4 Feature Extraction and Recognition of Iris

2.4.1 Convolutional Network Architecture

In this paper, a convolutional neural network architecture for iris feature extraction and recognition is proposed, which is suitable for FPGA deployment with small resources and low power consumption, as listed in Table 1. TensorFlow is used for training with 32-bit floating-point number accuracy. Moreover, the accuracy of training set and test set can reach up to more than 97%. Fig. 5 presents the accuracy and loss in the iteration process. Finally, the trained architecture is implemented on the hardware. In comprehensive consideration of speed and device resource, the pipelined architecture between different layers is used so as to make the arithmetical unit reused furthest [15]. Furthermore, the serial-parallel structure is used in different convolution kernels in the same convolutional layer and the fully connected layer in the last layer. Likewise, a large number of multipliers could be used in the convolutional layer and the fully connected layer. Thus, the approximate multiplier is applied in this paper for the approximate product calculation in order to reduce the resource consumption. The second power multiplier would be introduced and implemented in details in section 3.

Table 1. Convolutional neural network structure and parameters

Layer	Output	Parameter
Input	200×40×1	/
Conv1	198×38×6	168
Max_pooling1	66×12×6	/
Conv2	64×10×16	880
Max_pooling2	12×2×16	/
Flatten	1×1×384	/
Denseblock1	1×1×120	46200
Denseblock2	1×1×120	14520
Denseblock3	1×1×84	10164
Denseblock4	1×1×40	3400

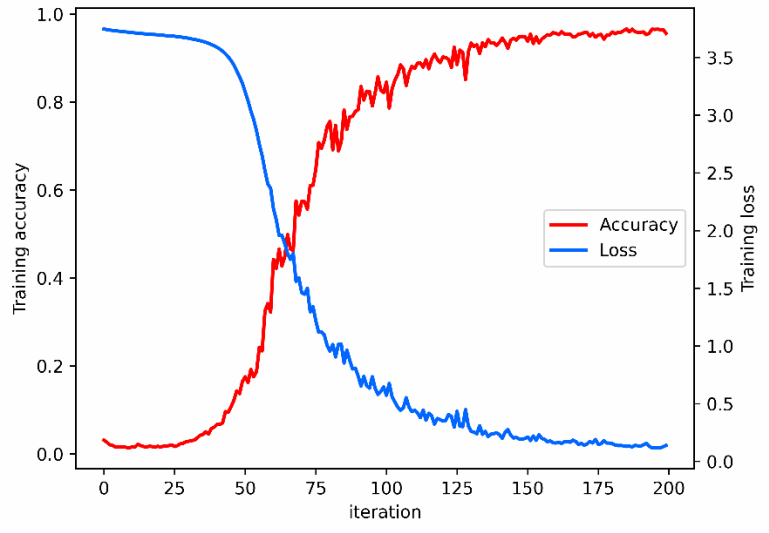


Fig. 5. Accuracy and loss curve

2.4.2 Hardware Implementation of Convolutional Layer

The size of the convolution kernel used in the convolution layer is 3×3, and the serial-parallel structure is used for the operation between different convolution kernels in the same convolutional layer so as to make a balance between speed and resource consumption [16]. The specific convolution process is shown in Fig. 6. The input feature and convolution kernel are extracted from ROM, respectively.

Simultaneously, two cache lines are used by the input feature for the data sliding, so that the 3×3 input feature matrix is obtained. A 3×3 weight matrix can also be obtained after the convolution kernel is taken out. At last, the input can be gotten by multiplying the weights by their corresponding input features and then summing their product up.

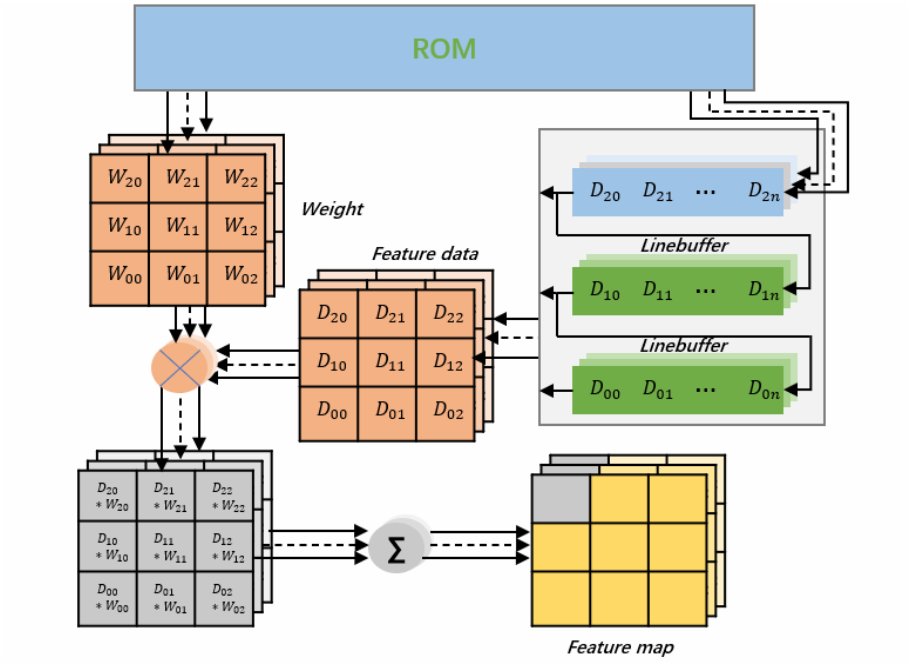


Fig. 6. Block diagram of the convolutional neural network

After the convolution operation, the activation function would be used to make the nonlinear factors added to its network. In this paper, the activation function used in the convolution layer is the ReLU function. The reasons for selecting the ReLU function as the activation function are as follows. ReLU function can improve the sparsity of the entire network and make the extracted features more representative. Moreover, its implementation on the hardware is simple, that is, only a selector is needed to implement the ReLU function performance. And it is very suitable for the implementation on FPGA with less resources due to its advantages of fewer resources consumption and low computational complexity.

2.4.3 Hardware Implementation of Pooling Layer

In this paper, the max-pooling is adopted in the pooling layer. The max-pooling of the pooling kernel is taken as an example, with a size of 3×3 and a step of 3, as exemplified in Fig. 7, in which the maximum value is found in each red box. The max-pooling is conducted on the output data stream of the convolutional layer. Clearly, every three rows and three columns are divided into a group. Taking the first group and the second group as an example, three data in the first row of each group are compared with each other successively by sliding, and the maximum value obtained is stored in the FIFO for the next comparison.

When the data slides to the second row, the maximum value of the three data in the second row is also firstly obtained, and then the corresponding data in the FIFO are compared with the maximum value to obtain a new maximum value, stored back into the FIFO. Likewise, as the data slide to the third row, a maximum value can be gotten. This maximum value is the result of maximum pooling, which is used for the subsequent convolutional layer or full connection layer operations.

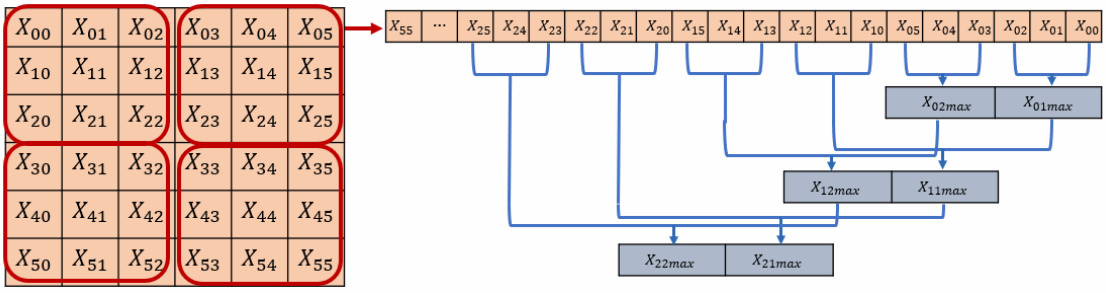


Fig. 7. Block diagram of the max-pooling layer network

2.4.4 Hardware Implementation of Fully Connected Layer

There is only one fully connected layer in this architecture, with 40 eigenvalues input and 40 eigenvalues output. Similarly, there is only one hidden layer with 40 neurons. The operation in each neuron is displayed in the formula of (6). Where x_i is the input of fully connected layer, W_i is the weight of each input feature in the neuron, and b is the bias of neurons. Finally, there still needs an activation function. In this paper, the Softmax function is used as the activation function. If the output in the fully connected layer adopts the full parallel calculation, a large number of multipliers would be consumed. This is absolutely inappropriate in a device with limited hardware resources. In consideration of the speed and resource consumption, the serial-parallel structure is used, shown in Fig. 8.

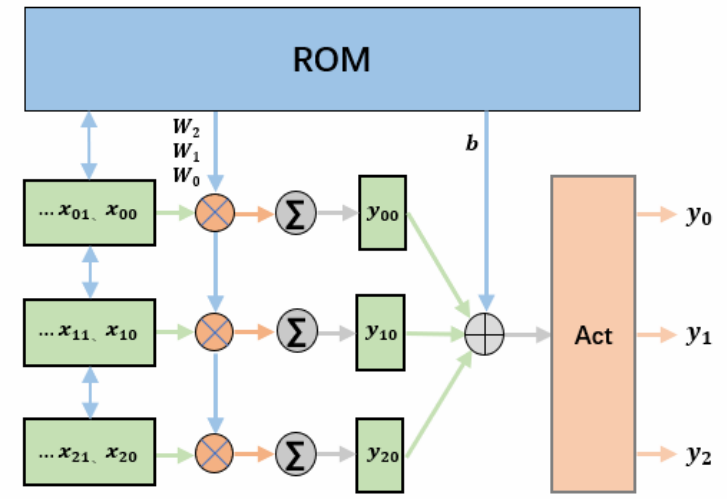


Fig. 8. Block diagram of the fully connected layer

$$y = b + \sum_{i=0}^{n} W_i \times x_i \tag{6}$$

The eigenvalue is divided into three equal parts, corresponding to three channels for parallel operation. Their weights are extracted from ROM. One eigenvalue accomplishes the calculation of three neurons at a time, and then the eigenvalue is stored in ROM for the next operation of three neurons, until all the neuron operations are completed.

Therefore, the fully connected layer only needs 9 multipliers, which reduces 200% of the multiplier resource cost in comparison with the fully parallel structure, and thus greatly reducing the burden of hardware resources. The results of the three channels are added together, and plus a bias, thus obtaining a final result. After the use of the activation function, the iris recognition results could be output.

The formula of Softmax activation function is shown in (7), which is implemented by exponential operation with a lookup table method.

$$p_i = \frac{e_i}{\sum_{j=0}^{n} e_j} \tag{7}$$

3 Approximate Multiplier

A large number of multipliers are applied in the convolutional layer, which exerts great pressure on the edge devices. The reason is that many edge devices do not have digital signal processing (DSP) units [5, 17]. This problem can be solved by the approximate multiplier proposed in this paper. Here, the design of the approximate multiplier is firstly introduced, and then the proposed approximate multiplier is evaluated and compared in many aspects.

3.1 Design of Approximate Multiplier

The approximate multiplier proposed in this paper is a power-of-two multiplier [18-19]. In other words, either the multiplier or the multiplicand is required to be a second power. The approximate multiplier is designed based on the principle of Booth 4 multiplier. Booth 4 multiplier halves the number of partial products by coding, and then the remaining partial products are added after corresponding shifting, thus obtaining the final result.

As can be seen from the above introduction, the Booth4 multiplier can provide faster speed but at the cost of consuming more hardware resources. In this paper, the proposed multiplier uses the coding manner of Booth4 multiplier to make the partial products halved with the assistance of marking signal *D* and conditioning signal *aju*.

This not only improves the speed but also greatly reduce the consumption of hardware resources. The specific coding is as shown in Table 2.

Table 2. Approximate multiplier encoding

b_{2k+1}	b_{2k}	b_{2k-1}	b_{2k+1} $(b_{2k+1})\,\&\,(!b_{2k})$	b_{2k} $(b_{2k})\,\&\,(!b_{2k-1})$	aju_{2k+1} $D_{2k+1}\,\&\,B_{MSB}$	aju_{2k} $D_{2k}\,\&\,B_{MSB}$	pp_k
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	1	0	B_{MSB}	A if $B_{MSB}=0$;!A if $B_{MSB}=1$
0	1	1	0	0	0	0	0
1	0	0	1	0	B_{MSB}	0	A<<1 if $B_{MSB}=0$;(!A) <<1 if $B_{MSB}=1$
1	0	1	1	0	B_{MSB}	0	A<<1 if $B_{MSB}=0$;(!A) <<1 if $B_{MSB}=1$
1	1	0	0	1	0	B_{MSB}	A if $B_{MSB}=0$;!A if $B_{MSB}=1$
1	1	1	0	0	0	0	0

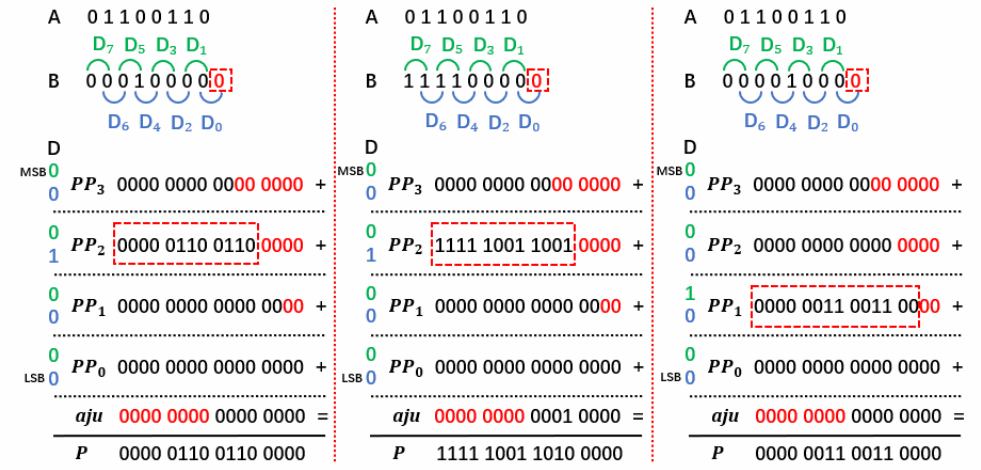
A $(a_n \ldots\ldots a_1 a_0)$ and B $(b_n \ldots\ldots b_1 b_0)$ are set as multiplicand and multiplier, respectively. Moreover, B is a second power, in which $b_{2k+1}, b_{2k}, b_{2k-1}$ (k = 0, 1, 2, ……, $\frac{n-1}{2}$, and $b_{-1}=0$) in B, are coded, and thus obtaining eight kinds of combinations in Table 2. The coding of marking signal D is obtained by formula (8), and partial product pp_k is operated according to D_{2k+1} and D_{2k} in Table 2. The conditioning signal aju_{2k} is worked out by formula (9), and it is mainly used to adjust the final result as B is negative. The final multiplicative result P is obtained by adding the conditioning signal and the sum of k partial products, as shown in formula (10).

$$D_n = (b_n)\,\&\,(!b_{n-1}) \tag{8}$$

$$aju_n = D_n\,\&\,B_{MSB} \tag{9}$$

$$P = aju_n + \sum_{0}^{k} PP_k \tag{10}$$

In order to introduce the design of approximate multiplier in details, three 8-bit second power multiplier are taken as an example, as shown in Fig. 9. The red *0* is a fixed value, and the fixed value *0* in pp_k aims to achieve the effect of shifting. The odd number bits (D_0 、 D_2 、 D_4 、 D_6) of marking signal D are the results of negation for the right ortho-position of the odd number bits (B_0 、 B_2 、 B_4 、 B_6). Likewise, the even number bits (D_1 、 D_3 、 D_5 、 D_7) of marking signal D are the results of negation for the right ortho-position of the odd number bits. The marking signals D obtained are divided into groups in pairs from low to high bit. In each group, D_{2k+1} and D_{2k} coding makes a one-to-one correspondence to the operation of partial product PP_k. As both D_{2k+1} and D_{2k} are 0, the partial product is 0. As shown in Fig. 9(a) and Fig. 9(b), D_5D_4 is "01". Thus, when the multiplier B is positive, PP_2 is the multiplicand A (the red dotted box shown in Fig. 9 (a)). When the multiplier B is negative, PP_2 is the negation of multiplicand A (the red dotted box shown in Fig. 9(b)). In Fig. 9(c), D_3D_2 is 10, and the multiplier B is positive. According to the coding table, the partial product PP_1 should be the value of the multiplicand A after shifting one bit to the left (the red dotted box shown in Fig. 9 (c)). Finally, all the partial product and conditioning signals are added together to get the result P of 8-bit second power multiplication.



(a) (b) (c)

Fig. 9. Examples of the 8-bit approximate multiplier

In the second power multiplication, only one partial product at most could affect the final result, such as PP_2 in Fig. 9(a) and Fig. 9(b), PP_1 in Fig. 9(c), etc. The rest of the product is 0. Therefore, there are two schemes to make the accumulation of partial product. For the first scheme, the adder is directly used to complete the accumulative operation of final partial product. Differently, for the second scheme, the selector is used to select the non-zero partial product.

It can be concluded by analyzing the resource consumption, delay and power consumption of the two schemes that the first scheme is optimal.

3.2 Evaluation of Approximate Multiplier

The approximate multiplier is developed in Verilog with the primitive. The 8×8 and 16×16 approximate multipliers are deployed to Intel FPGA Cyclone 10LP 10CL025 device which is targeted on the Intel Cyclone 10 LP FPGA Evaluation Kit. It should be noted that the key parameters from the approximate multiplier are mostly based on 8×8 multiplier in the past researches. Hence, the 8×8 multiplier is used for parameter comparison, while the 16×16 multiplier is deployed on CNN. The deployment of 16×16 multiplier is compared with that of 8×8 multiplier, and the results are listed in Table 3. The approximate

multiplier proposed in this paper is designed after the balance among the area, performance, and power consumption. The design is not optimal under the single dimension of area, performance, and power consumption, respectively. However, the proposed approximate multiplier is optimal under the comprehensive index latency energy production (LEP) [22]. The error analysis after deployment is listed in Table 4 and obviously, the proposed approximate multiplier has the best comprehensive performance.

Table 3. Implementation results of different multipliers

Multiplier Architectures	8×8				16×16			
	LEs	Latency (ns)	E, (pJ)	LEP	LEs	Latency (ns)	E, (pJ)	LEP
Proposed	103	8.86	25.45	225.49	335	19.04	67.13	1278.16
LE-based IP	154	11.10	29.20	324.12	467	22.12	100.6	2225.27
[19]	116	8.30	27.20	225.76	399	20.68	69.10	1428.99
[20]	120	5.00	/	/	350	16.56	/	/
[21]	123	5.71	/	/	362	17.44	/	/

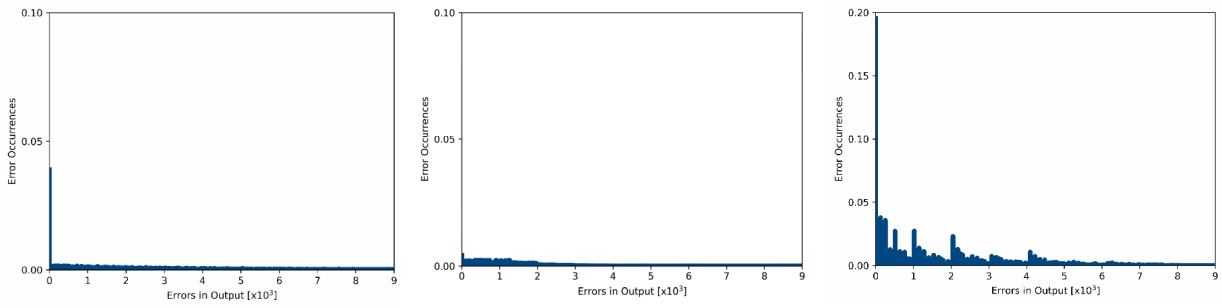
Table 4. Different types of error analysis of 8×8 multipliers

Error Type	Multiplier Architectures			
	Proposed	[23]	[24]	[25]
Maximum Error Magnitude	8170	8288	7225	14450
Average Error	1293.544	1592.265	1354.687	903.125
Average Relative Error	0.123	0.129	0.144	0.032
Error Occurrences	35712	52731	53375	30625

The effect of the final deployment of the 16×16 approximate multiplier on the operation results is presented more intuitively in two aspects. On the one hand, the proportion of Error Occurrences is compared, and the results are shown in Fig. 10. Here, the proportion of Error Occurrences is a well-adopted quality metric [21] and [23]. It is found that the proportion of Error Occurrences of the proposed approximate multiplier is more superior to that of the approximate multiplier of open source. On the other hand, the IP-based 16×16 multiplier and the designed 16×16 approximate multiplier in this paper perform the mean filtering, respectively, to smooth images. Then, the effect is shown in Fig. 11. It can be concluded by comparison that the filtering effect of second power multiplier is similar to that of the common multiplier.

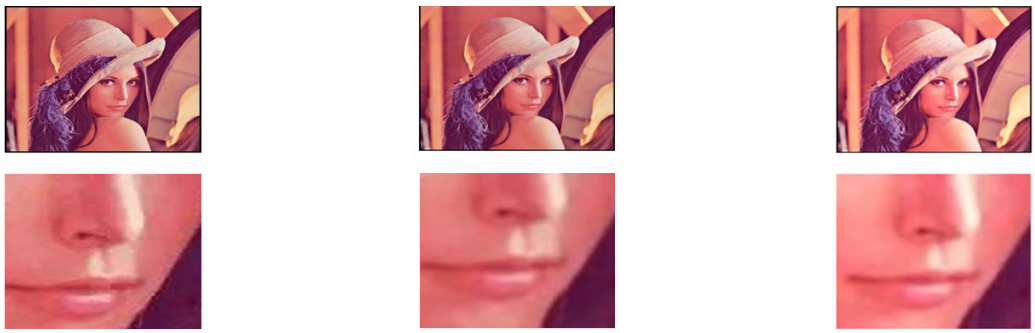
4 Experiments and Analysis

As an FPGA-based platform for iris recognition system implementation, we utilized the Intel Evaluation Kit with Intel Cyclone 10LP FPGA. To meet low power consumption, this chip uses a 20nm CMOS technological with 25K LUTs, 594 Kbit M9K Memory, 66 18x18 Multipliers, and 4 PLLs. Table 5 shows the resource consumption of this design. It does not use precious DSP units with multiplier resources, as design approximate multipliers are effective and low LE resource consumption.



(a) proposed (b) [21] (c) [23]

Fig. 10. Proportion of Error effects of different approximate multipliers



(a) raw picture (b) filtering effect based on LE-based multiplier IP (c) filtering effect based on approximate multiplier

Fig. 11. Examples of the filtering effect based on different multiplier

Table 5. Resource and performance analysis of system

Resource & Performance	Consumption with approximate multipliers	Consumption with DSP unit	Consumption with multiplier IP
LEs	9,498	6,483	11,389
Registers	3,090	3,000	3,480
Memory (Kb)	49.20	40.60	46.15
DSP (Embedded Multiplier)	0	9	0
Max Freq (MHz)	200	220	178
Latency (ms)	7.613	6.980	8.048
Power (mW)	84.57	90.86	90.21
Accuracy (%)	96.88	97.62	97.62

In order to further evaluate the performance of the proposed iris recognition system, several common embedded development platforms (Raspberry Pi 3B+, Rockchip RK3399 Pro, and Jetson Nano) were selected for comparison. The Raspberry Pi comes with a 1.2 GHz Cortex-A53 core. Rockchip RK3399 Pro uses Cortex-A72 as its processor core and 2.0GHz as its main frequency. It is packed with a special NPU module for machine learning. Jetson Nano is cortex-A57 as the core, the main frequency is 1.43GHz, with a 128-core NVIDIA Maxwell based GPU. On Raspberry Pi, we used the TensorFlow Lite to perform system. On the Rockchip platform, the RK-NN-Toolkit tool is used to realize CNN parallel processing [26]. In order to perform parallel processing on the GPU, we used PyCuda [27] on the Jetson Nano platform to achieve parallel processing of CNN. Table 6 shows the comparison results for all platforms. In terms of computing performance, Jetson Nano is the best among all platforms due to the GPU's high frequency parallel processing capability. But in a embedded system, power consumption is important too. The power latency production (PLP) can effectively evaluate the comprehensive performance of the platform. It can be found that our proposed FPGA-based system achieves the best overall performance. Specifically, the proposed system better than 48 times, 19 times, and 11 times in PLP with Raspberry Pi, Rockchip RK3399 Pro, and Jetson Nano respectively.

Table 6. Performance analysis of different embedded system platforms

Platform	Processor	Clock (MHz)	Lantency (ms)	Power (mW)	PLP
Raspberry Pi 3B+ (baseline)	Cortex-A53	1200	12.737	2440	31078.28
Rockchip RK3399 Pro	Cortex-A72 with NPU	2000	3.532	3610	12750.52
Jetson Nano	Cortex-A57 with Maxwell-based GPU	1430	2.277	3190	7263.63
Proposed	Cyclone 10LP FPGA	200	7.613	84.57	643.83

5 Conclusion

This paper presents a safe and portable identity recognition system during the pandemic — an Iris recognition system based on FPGA. It doesn't require users to take off their masks while recognizing, and is more comprehensive than past embedded system deployments, especially in its performance and power consumption. After deploying the preprocessing module onto the FPGA, we further deployed CNN for feature extraction and classification. Moreover, we designed an approximate multiplier to complete multiplication calculations with less power and DSP resource consumption, with no significant loss in recognition accuracy. Experimental results show that the recognition speed of the proposed system increased 40% compared with Raspberry Pi. And PLP is 18 times better than Rockchip RK3399 Pro, 11 times better than Jetson Nano.

For future researches, we focus on the further evaluation and deployment of this system. Devices with lower power consumption can be adopted, such as Lattice. We can also introduce FPGA with no DSP resource at all, and compare its LE-based multiplier IP with our design. Finally, in order to achieve lower power and resource consumption as well as better deployment on the terminal, a more lightweight CNN can be realized while ensuring accuracy and speed.

Acknowledgements

The authors would like to thank VeriMake Innovation Lab for providing the equipment for development and test.

References

[1] C.G. Schwarz, W.K. Kremers, T.M. Therneau, R.R. Sharp, J.L. Gunter, P. Vemuri, A. Arani, A.J. Spychalla, K. Kantarci, D.S. Knopman, R.C. Petersen, C.R. Jack, M.C. Rochester, Identification of anonymous MRI research participants with face-recognition software, New England Journal of Medicine 381(17)(2019) 1684-1686.

[2] S. Aleem, P. Yang, S. Masood, P. Li, B. Sheng, An accurate multi-modal biometric identification system for person identification via fusion of face and finger print. World Wide Web 23(2)(2020) 1299-1317.

[3] K. Okereafor, I. Ekong, I.O. Markson, K. Enwere, Fingerprint Biometric System Hygiene and the Risk of COVID-19 Transmission, JMIR Biomedical Engineering 5.1(2020) e19623.

[4] M. Roukhami, M.T. Lazarescu, F. Gregoretti, Y. Lahbib, A. Mami, Very Low Power Neural Network FPGA Accelerators for Tag-Less Remote Person Identification Using Capacitive Sensors, IEEE Access 7(2019) 102217-102231.

[5] F. Ge, N. Wu, H. Xiao, Y. Zhang, F. Zhou, Compact Convolutional Neural Network Accelerator for IoT Endpoint SoC, Electronics 8(2019) 497.

[6] J. Daugman, How iris recognition works, IEEE Transactions on Circuits & Systems for Video Technology 14(1)(2004) 21-30.

[7] H. Ngo, R. Rakvic, R. Broussard, R. Ives, M. Carothers, Architecture Design for Feature Extraction and Template Matching in a Real-Time Iris Recognition System. Electronics 10(3)(2021) 241.

[8] Institute of Automation, Chinese Academy of Sciences, CASIA Iris Image Database. <http://biometrics.idealtest.org/> 2020 (accessed 21.05.01).

[9] D. Yang, M. Ren, B. Xu, Retinal blood vessel segmentation with improved convolutional neural networks, Journal of Medical Imaging and Health Informatics 9(6)(2019) 1112-1118.

[10] Y. Li, W. Hu, H. Dong, X. Zhang, Building damage detection from post-event aerial imagery using single shot multibox detector, Applied Sciences 9(6)(2019) 1128.

[11] F. Boutros, N. Damer, K. Raja, R. Ramachandra, F. Kirchbuchner, A. Kuijper, Iris and periocular biometrics for head mounted displays: Segmentation, recognition, and synthetic data generation, Image and Vision Computing 104(2020) 104007.

[12] A. Noruzi, M. Mahlouj, A. Shahidinejad, Iris recognition in unconstrained environment on graphic processing units with CUDA, Artificial Intelligence Review 53(1)(2020) 1-25.

[13] D. Sangeetha, P. Deepa, FPGA implementation of cost-effective robust Canny edge detection algorithm, Journal of Real-Time Image Processing 16(4)(2019) 957-970.

[14] S. Ganorkar, M. Memane, Iris recognition using discrete wavelet transform, International Journal of Advances in Engineering & Technology (2012) 147-151.

[15] S. Colleman, V. Marian, High-Utilization, High-Flexibility Depth-First CNN Coprocessor for Image Pixel Processing on FPGA, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 29(3)(2021) 461-471

[16] N. Attaran, A. Puranik, J. Brooks, T. Mohsenin, Embedded Low-Power Processor for Personalized Stress Detection, IEEE Transactions on Circuits & Systems II Express Briefs (2018) 1-1

[17] L. Grzymkowski, T.P. Stefański, Performance Analysis of Convolutional Neural Networks on Embedded Systems, in: Proc. 2020 27th International Conference on Mixed Design of Integrated Circuits and System (MIXDES), 2020

[18] X. Xu, Q. Lu, T. Wang, J. Liu, Y. Hu, Y. Shi, Efficient hardware implementation of cellular neural networks with powers-of-two based incremental quantization, in: Proc. 2017 Proceedings of the Neuromorphic Computing Symposium, 2017.

[19] S. Perri, F. Spagnolo, F. Frustaci, P. Corsonello, Parallel architecture of power-of-two multipliers for FPGAs, IET Circuits, Devices & Systems 14(3)(2020) 381-389

[20] R.W. Griffith, Area-Efficient Fast Truncated 8x8 Soft Multipliers for Field Programmable Gate Array Accelerators, [Doctoral dissertation] UCLA, 2020.

[21] K.K. Senthilkumar, K. Kumarasamy, V. Dhandapani, Approximate Multipliers Using Bio-Inspired Algorithm, Journal of Electrical Engineering & Technology 16(1)(2021) 559-568.

[22] S. Ullah, T.D.A. Nguyen, A. Kumar, Energy-Efficient Low-Latency Signed Multiplier for FPGA-Based Hardware Accelerators, IEEE Embedded Systems Letters 15(5)(2020) 1.

[23] S. Ullah, S. Rehman, B.S. Prabakaran, F. Kriebel, M.A. Hanif, M. Shafique, A. Kumar, Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators, in: Proceedings of the 55th Annual Design Automation Conference, 2018.

[24] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, J. Henkel, Architectural-space exploration of approximate multipliers, in: Proc. 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016.

[25] P. Kulkarni, P. Gupta, M. Ercegovac, Trading accuracy for power with an underdesigned multiplier architecture, in: Proc. 2011 24th International Conference on VLSI Design, 2011.

[26] H. Lan, J. Meng, C. Hundt, B. Schmidt, M. Deng, X. Wang, W. Liu, Y. Qiao, S. Feng, FeatherCNN: Fast Inference Computation with TensorGEMM on ARM Architectures, IEEE Transactions on Parallel and Distributed Systems 31(3)(2019) 580-594.

[27] B. Calabrese, R. Velázquez, C. Del-Valle-Soto, R.D. Fazio, N.I. Giannoccaro, P. Visconti, Solar-Powered Deep Learning-Based Recognition System of Daily Used Objects and Human Faces for Assistance of the Visually Impaired, Energies 13(22)(2020) 6104.