Design of Contactless Intelligent Epidemic Prevention System based on PYNQ

Wang Lingzhi, Wang Jianyuan, and Xu Yingjing

Abstract—**To combat the COVID-19 virus, extensive studies are being conducted on contactless alternative methods for measuring body temperature and detecting whether masks are being worn. This study investigated a system that employed ZYNQ-7020 as the main controller. The accurate non-contact measurement of a persons body temperature was realized by the calibration and compensation of infrared sensor data collected by an MLX90614 thermopile. In addition, OpenCV and a convolutional neural network (CNN) algorithm were used for face image recognition and to detect whether a person was wearing a mask. A mask detection algorithm was developed based on high-level synthesis and implemented by Python productivity for Zynq (PYNQ). The PYNQ test results were output through a high-definition multimedia interface. Users could also view the real-time human body temperature and face image recognition results through the web. The system achieved a temperature measurement distance of 30 cm, with a measurement error of ± 0.3 °C and measurement range of 2560 °C. The system also had an alarm function with a buzzer that could be used for an over-temperature alert. The algorithm used the characteristics of field programmable gate array parallel computing to improve the CNN calculation in image processing. A variety of optimization strategies were implemented to achieve hardware acceleration, which improved the mask recognition rate.**

Index Terms—**Contactless intelligent epidemic prevention system, Convolutional neural networks,Infrared temperature measurement, Mask detection, ZYNQ-7020.**

I. Introduction

COVID-19 is mainly transmitted directly by the transmission of droplets when a patient sneezes, coughs, or speaks, and indirectly by contacting an object with droplets on its surface, which then come into contact with the mouth, nose, or eyes, leading to infection [1], [2], [3]. Therefore, it is particularly important to detect whether people are wearing masks and measure their body temperatures in crowded public places such as train stations and airports to prevent suspected patients from entering and control the spread of the virus [4], [5].Among the traditional detection methods, manual detection or surveillance is most commonly used [6], [7], [8], [9]. However, this traditional method consumes considerable human and material resources. In this practical application context, there is an urgent need for a contactless intelligent epidemic prevention system that can satisfy the need for the rapid and mass unmanned and contactless detection of target persons and mask wearing. In scenarios in which it is not possible to deploy servers and there is sensitivity to cost, a large amount of computation for face detection and mask detection needs to be performed on the end device, which requires an implementation platform with high performance and parallel computation [10]. In this study, a ZYNQ-7020 was chosen as the main controller to drive an OV5640 camera to capture the target image. The recognition of face images and the detection of mask wearing were implemented through OpenCV and convolutional neural network (CNN) algorithms [11], [12]. The mask detection algorithm was developed based on high-level synthesis (HLS) and implemented by Python productivity for Zynq (PYNQ) [13]. The CNN computation during image processing was accelerated using the feature of field programmable gate array (FPGA) parallel computing. The detection results were output via a high-definition multimedia interface (HDMI). The system determined a persons body temperature through the calibration and compensation of infrared (IR) sensor data acquired using an MLX90614 thermopile, which had a temperature measurement distance of up to 30 cm, measurement error of ± 0.3 °C, and measurement range of 2560 °C. An over-temperature alarm function was also incorporated. In addition, a hypertext transfer protocol (HTTP) server was deployed on ZYNQ, which would allow users to view real-time human body temperature and face picture recognition results through a web page. A schematic of the system is shown in Fig. 1.

Manuscript received April 19, 2021; revised March 16, 2022; This research was sponsored by Education Reform Project of Fujian Provincial under Grant FBJG20200049; Natural Science Foundation Project of Fujian Province under Grant 2020J05169; Teaching reform of Minnan Normal University under Grant JG201918.

Wang Lingzhi is an Associate Professor of Department of Electronic and Information Engineering, Xiamen City University, CO 361008, China. (corresponding author phone: +86 13515957743; e-mail: 64564254@qq.com)

Wang Jianyuan is an undergraduate of the College of Physics and Electronic Information Engineering, Minnan Normal University, CO 363000, China. (e-mail: wwangjiangyuan@163.com)

Xu Yingjing is an undergraduate of the College of Physics and Electronic Information Engineering, Minnan Normal University, CO 363000, China. (e-mail: xuyingjing@163.com)

II. General Framework of the System

ZYNQ is a new generation of fully programmable systems-on-a-chip proposed by Xilinx, combining FPGA logic components (the programmable logic (PL) part) and a dual-core ARM Cortex-A9 processor (the processing subsystem (PS) part), with the hardware and software working together. In this design, an xc7z020clg400-2 (referred to as the ZYNQ-7020) was selected as the main controller for the entire system. The AXI bus protocol was used as the bridge for PL/PS data signal communication in the ZYNQ-7020 system architecture. Fig. 2 shows the overall framework of the contactless intelligent epidemic prevention system. In the PL, the Vivado HLS tool was used to implement the design of IP cores for camera image data acquisition, the HDMI display driver, the MLX90614 sensor driver, and the CNN. The PS part was based on the PYNQ open source framework, which mainly included the design of the image storage, face detection and algorithm optimization, HTTP server construction, and design of the overall functional logic.

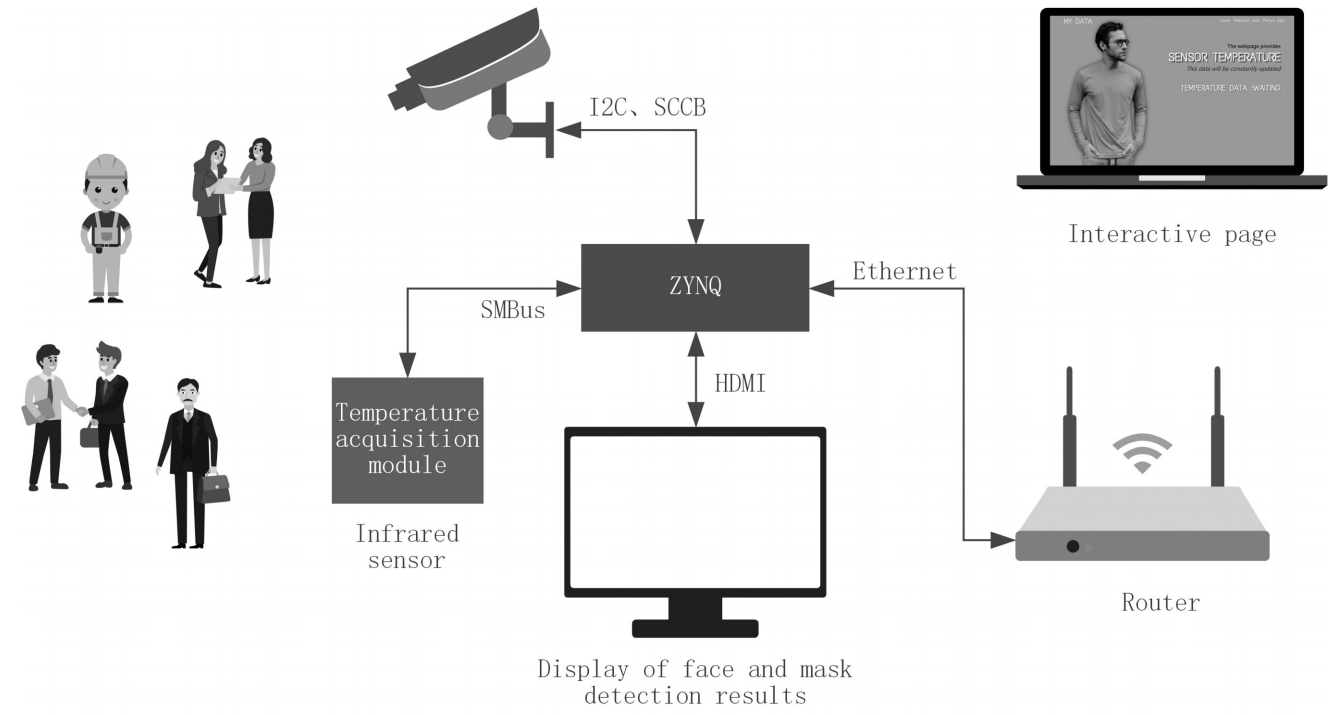

Fig. 1. Schematic of the contactless intelligent epidemic prevention system.

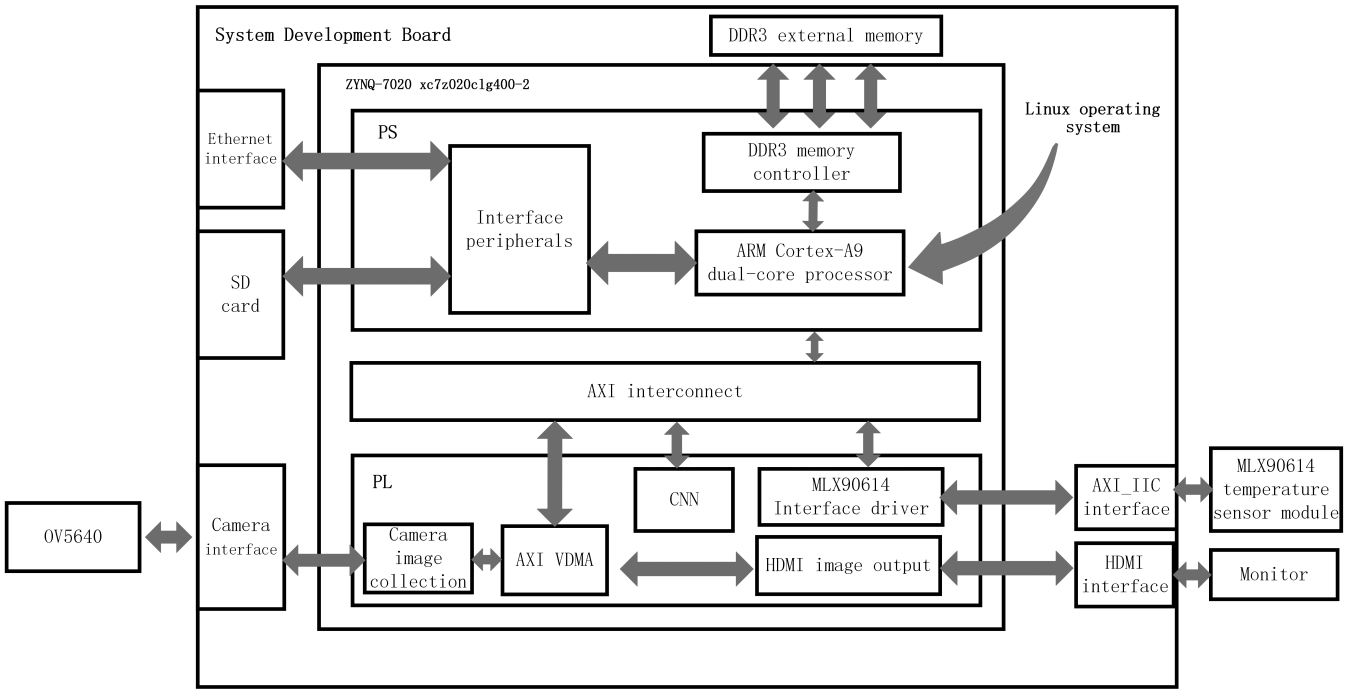

Fig. 2. The overall framework of the system.

III. VIVADO HLS-BASED PL SYSTEM DESIGN

A. Introduction to Vivado HLS

In the traditional FPGA development flow, which requires considerable hardware-oriented design work and verification time, Xilinx has introduced the Vivado HLS tool to shorten the development cycle, which allows the for direct development of the Xilinx family of FPGAs using C, C++, or System C. Vivado HLS has a large number of optimized instructions, which significantly reduces the time traditionally spent on FPGA development using RTL descriptions. In this design, the Xilinx Vivado HLS development tool was used to package the camera image acquisition, IR temperature measurement, and CNN into IP cores that could be called by the PYNQ framework.

B. OV5640 Image Acquisition

The system used an OV5640 camera for image acquisition. The simplified framework is shown in Fig. 3. The ARM processor performed an initial configuration of the OV5640 camera via the serial camera control bus to enable it to output

video data at a set resolution. Then, the Video to AXI4-Stream IP core was used to convert the captured video data stream to the AXI4-Stream data format required for virtual direct memory access (VDMA). VDMA connected to the AXI HP port in coordination with the AXI Smartconnect IP core to efficiently access DDR3 and write the image data into the DD3 external memory. At the same time, the processor processed the cached image data for face detection, recognition, etc. [14], [15], [16].

After the recognition and detection of the image on the PS end, VDMA read the processed image data from the DDR3 and transmitted it to the AXI4-Stream to Video Out IP core. The AXI4-Stream to Video Out IP core, under the control of the VTC IP core, converted the AXI4-Stream data into the video output data format, transmitted the output video data stream to the DVI Transmitter IP core, and finally displayed the recognition results using the LCD.

C. IP Core Design for IR Temperature and Distance Measurement

A Melexis MLX90614ESF-DCI high-precision IR temperature sensor was used for the IR temperature measurement acquisition [17], [18]. The IR sensor communicated with the ZYNQ through the SMBus bus protocol, and the connection is shown in Fig. 4. To ensure that the SMBus bus data transfer remained at a high electrical level in the idle state, two 4.7 kΩ pull-up resistors were connected, and a filter capacitor (C2) with a capacitance of 0.1 F was placed between the ends of the sensor power supply.

The sensor output data were linearly proportional to the object temperature and had a wide temperature range. This design initiated and controlled the conversion of the sensors data and read the converted temperature data from its internal read only memory. The thermopile sensor output signal is given by Eq. (1):

$$V_{ir}(T_a, T_o) = A \times (T_o^4 - T_a^4) \tag{1}$$

where T_o is the target temperature (K), T_a is the ambient temperature (K), and A is the sensor sensitivity constant.

The IR distance detection circuit is implemented using an LM393 comparator with a pair of it transmitter-receiver tubes. The circuit takes advantage of the IR receiver tube's sensitivity to IR light to achieve the distance detection requirements in this design. The detection circuit is shown in Fig. 5, in which LM393 pin 6 is connected to an adjustable potentiometer that adjusts the threshold voltage, whose magnitude is adjusted according to the actual distance detection needs.

where T_o is the target temperature (K), T_a is the ambient temperature (K), and A is the sensors sensitivity constant. The IR distance detection circuit was implemented using an LM393 comparator with a pair of IR transmitterreceiver tubes. The circuit took advantage of the IR receiver tubes sensitivity to IR light to achieve the distance detection requirements in this design. The detection circuit is shown in Fig. 5, in which LM393 pin 6 is connected to an adjustable potentiometer that adjusts the threshold voltage according to the actual distance detection needs.

Because the IR light generated by the IR emitter in the IR distance measurement module could affect the accuracy of the temperature acquisition, triodes were used to design a distance detection module control circuit, as shown in Fig. 6. When the ctrl outputs a high electrical level, triode Q1 conducts, and the IR distance detection module is in a working state; when the ctrl outputs a low electrical level, triode Q1 does not conduct, and the IR distance detection module does not work or emit IR light, thus avoiding the effect on the temperature measurement part.

D. Mask-Wearing CNN IP Core Design

The mask wearing detection classified the face images in the face detection results into two categories: "with mask" and "without mask." The design was based on a modified CNN model structure, and the CNN acceleration IP kernel was designed to detect the images of faces and label the detected images with the detection results [19].

Preparation of dataset

This design mainly used the CNN to achieve mask wearing detection. It was first necessary to train the CNN model, for which a RMFD mask shielding face dataset was used [20], [21]. This dataset contains simulated mask-wearing face data of 10,000 people and 500,000 faces. The design randomly selected 24,000 images from the dataset for training: 16,800 for the training set and 7,200 for the test set. The images in the datasets were preprocessed in a sequence that included image grayscale conversion, image size adjustment, and image file renaming, to generate a 28 $\times$ 28 dataset in the IDX format.

CNN model

This design used a CNN model structure based on a modified CNN, as shown in Fig. 7. The input to the model was a 28 $\times$ 28 $\times$ 1 single-channel grayscale map, and the output was a 1 $\times$ 2 matrix with values indicating the probability of the corresponding class of the input image through two convolutional layers, two max pooling layers, and two fully connected layers. The network was trained using a script. After the training was completed, the model was tested using the test set and had an accuracy of approximately 99.9%. The parameters from the training were saved to be used when the network is deployed.

The network is trained using a script. After the training is completed, the model is tested using the test set with an accuracy of approximately 99.9 %. The parameters from the training are saved to be used when the network is deployed.

E. Parallel Analysis and Acceleration Methods for Mask Detection

Wearing detection algorithms

To accelerate the mask wearing detection analysis, the mask wearing detection function of this design was implemented by the CNN, which required a large number of convolutional and pooling operations. These were the most time consuming processes of the forward computation process of the CNN. Because the convolutional operation process had the characteristics of parallel computing, the hardware parallel acceleration of this process was performed at the PL end, and the speed of the entire process was

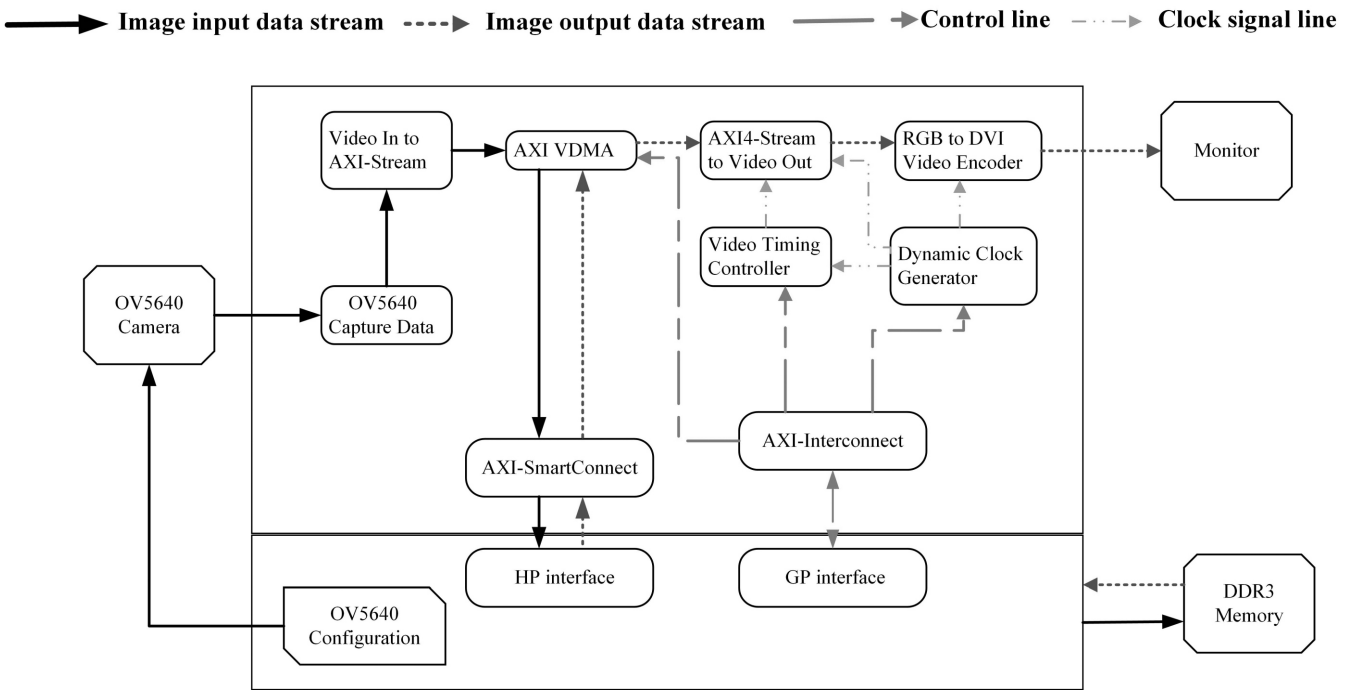

Fig. 3. Image acquisition and display frame diagram.

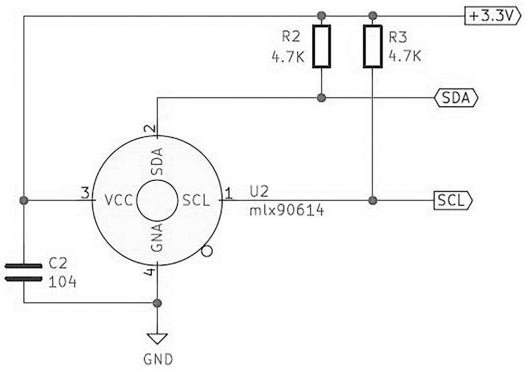

Fig. 4. IR temperature sensor acquisition circuit.

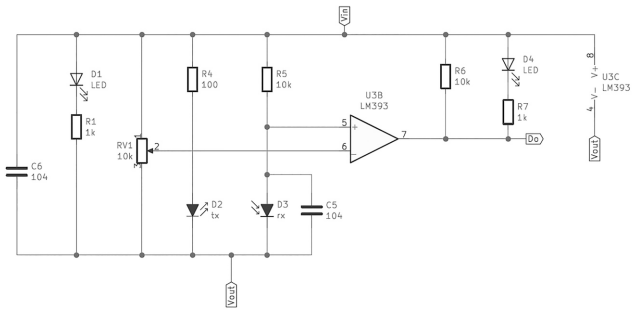

Fig. 5. IR distance detection circuit.

considerably increased by optimization strategies such as loop reconstruction, array division, loop expansion, and data access [22].

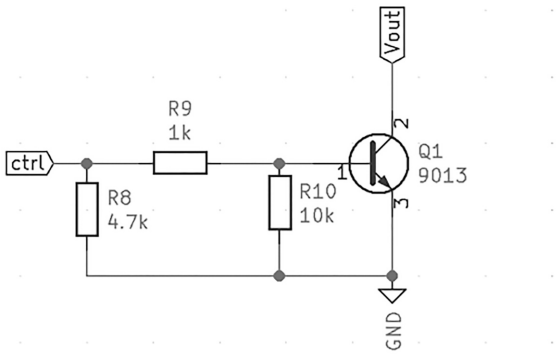

Fig. 6. Distance detection of control circuit.

Acceleration of convolution process on FPGA

The flow of the convolutional operation is shown in Fig. 8. The inputs of the convolutional operation were CHin feature maps, and the outputs were CHout feature maps with a size of C × R. Each output feature map was obtained by convolving the convolution kernel with the respective input features. Thus, there were CHout × CHin convolution kernels with a size of K × K.

The convolution formula is shown in Eq. (2):

$$Out[cho][r][c] = \sum_{chi=0}^{CHin-1} \sum_{kr=0}^{K-1} \sum_{kc=0}^{K-1} In[chi] [r+kr][c+kr] \times W[cho][chi][kr][kc] \quad (2)$$

where Out denotes the output of the convolution result; cho and chi denote the output and input channels, respectively; r denotes the row of the output; c denotes the column of the output; In denotes the input of the convolution; and kr, kc, and W denote the row, column, and weight matrix of

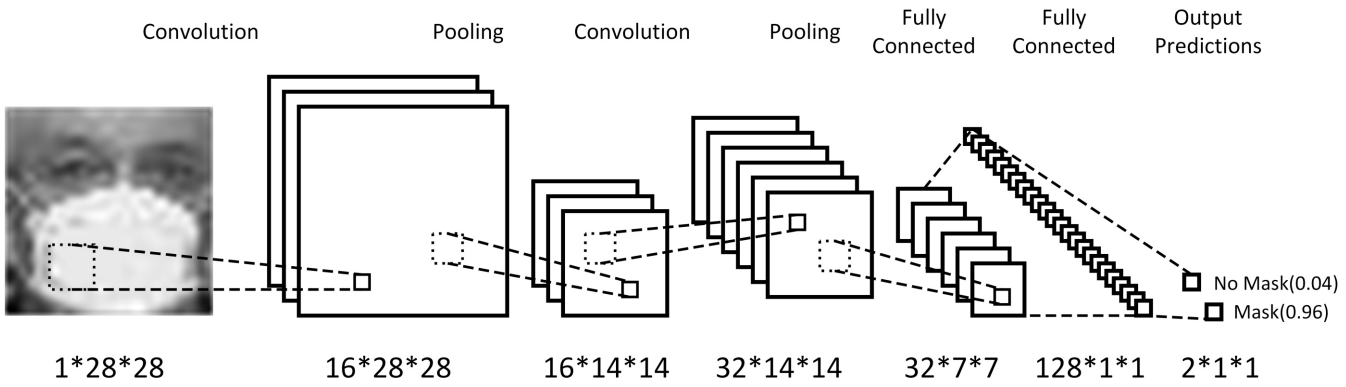

Fig. 7. Structure of the network model used in this design.

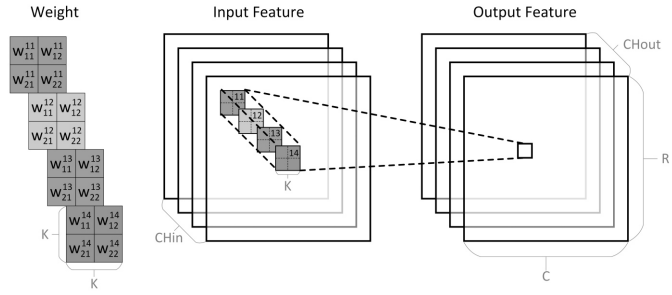

Fig. 8. Schematic diagram of the convolutional operation.

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1060833	1060833	1060833	1060833	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Output_Channel	1060832	1060832	66302	-	-	16	no
+ Row	66300	66300	2550	-	-	26	no
++ Col	2548	2548	98	-	-	26	no
+++ Kernel_Row	96	96	32	-	-	3	no
++++ Kernel_Col	30	30	10	-	-	3	no

Fig. 9. Vivado HLS synthesis report.

the convolution kernel, respectively [23]. The corresponding HLS code is shown as follows.

1. for (int $cho = 0; cho < CHout; cho++$)
2. for (int $chi = 0; chi < CHin; chi++$)
3. for (int $r = 0; r < Rout; r++$)
4. for (int $c = 0; c < Cout; c++$)
5. for (int $kr = 0; kr < K; kr++$)
6. for (int $kc = 0; kc < K; kc++$)
7. $Out[cho][r][c] += In[chi][r + kr][c + kc] * W[cho][chi][kr][kc]$;

The network layer parameters used are listed in Table I.

TABLE I
NETWORK LAYER PARAMETERS

CHin	Rin	Cin	CHout	Rout	Cout	K
1	28	28	16	26	26	3

The input, output, and weight variables in the test code were the single-precision floating-point type, and the Vivado HLS synthesis when the hardware period was set to 10 ns is reported in Fig. 9. It can be seen that the latency reached 1,060,833 clock cycles because the loop in the designed accelerator did not perform a pipelined operation.

The design took advantage of the parallel computing features of FPGAs to adopt various optimization strategies for the loop body in the convolutional code, with the following specific steps.

Loop reconstruction

Because the hardware acceleration process of the convolution was parallelized for both the input and output channels, the hardware accelerator needed to be repeatedly called during the convolutional operation. Thus, it was necessary to put the loops of the input and output channels in the innermost loop of the entire convolutional operation. Because the multiplicative accumulation of the convolution satisfied the associative law, simply changing the order of the loops was sufficient, and the modified code after the loop refactoring is as follows:

1. for (int $r = 0; r < Rout; r++$)
2. for (int $c = 0; c < Cout; c++$)
3. for (int $kr = 0; kr < K; kr++$)
4. for (int $kc = 0; kc < K; kc++$)
5. for (int $cho = 0; cho < CHout; cho++$)
6. for (int $chi = 0; chi < CHin; chi++$)
7. $Out[cho][r][c] += In[chi][r + kr][c + kc] * W[cho][chi][kr][kc]$;

Division of arrays

Hardware parallel optimization in convolutional computing requires parallel access to inputs, outputs, and weights. They can be expanded by their channels and divided into multiple random access memory (RAM) blocks, which allows the data in the RAM blocks to be read in parallel to reduce latency when performing parallel computations. Vivado HLS provides the ARRAY_PARTITION constraint instruction to perform array partitioning. The modified code

after array partitioning is as follows:

1. float $In[CHin][Rin][Cin]$;
2. $\#pragmaHLSARRAY\_PARTITIONvariable = Incompletedim = 1$
3. float$Out[CHout][Rout][Cout]$;
4. $\#pragmaHLSARRAY\_PARTITIONvariable = Outcompletedim = 1$
5. float$W[CHout][CHin][K][K]$;
6. $\#pragmaHLSARRAY\_PARTITIONvariable = Wcompletedim = 1$
7. $\#pragmaHLSARRAY\_PARTITIONvariable = Wcompletedim = 2$

The Vivado HLS synthesis report showed a latency of 899,809 clock cycles, and thus a reduction of 97,344 clock cycles compared to this number before array partitioning.

Loop unrolling

The input and output channels of the convolutional operations needed to be optimized in parallel and be fully unrolled in the loop to perform hardware parallel acceleration optimization. Vivado HLS provides the UNROLL instruction to unroll the loop, and the modified code after the loop is unrolled is as follows:

1. for (int $cho = 0; cho < CHout; cho++$)
2. $\#pragmaHLSUNROLL$
3. for (int $chi = 0; chi < CHin; chi++$)
4. $\#pragmaHLSUNROLL$
5. $Out[cho][r][c] += In[chi][r + kr][c + kc] * W[cho][chi][kr][kc]$;

The report after the Vivado HLS synthesis showed a latency of 66,301 clock cycles, which was a reduction of 833,508 clock cycles compared to the number before the loop was unrolled. At this point, the unrolled loop body was not pipelined.

Pipelining

After the unrolling process, the loop was not pipelined yet, which indicated the need for further optimization. To pipeline-unroll the loop body for both the input and output channels, Vivado HLS provides the PIPELINE instruction to unroll and pipeline the loop body with the following modified code:

1. $\#pragmaHLSPIPELINE$
2. for (int $cho = 0; cho < CHout; cho++$)
3. for (int $chi = 0; chi < CHin; chi++$)
4. $Out[cho][r][c] += In[chi][r + kr][c + kc] * W[cho][chi][kr][kc]$;

The Vivado HLS synthesis is reported in Fig. 10, in which it is observed that the indicator Pipelined is marked as yes, indicating that the loop body has been pipelined. In addition, there was a reduction of 23,703 clock cycles compared to the case before pipelining.

However, the reported indicator Initiation Interval value was not 1. Thus, it is clear that the Out array was involved in self-adding operations during the loop. As shown in Fig. 11, the Out array was read, computed, and written during the single loop interval. Thus, the Out array was not read by the next loop when the current loop was not finished.

To solve the Initiation Interval problem, we further optimized the code and put the row and column of the convolution kernel into the outermost layer of the convolution operation loop (see Fig. 12). The two adjacent loop intervals accessed the data at different positions of the Out array. There was no data dependency relationship, which could be implemented by pipelining. The code read the Out array after an R * C cycle interval, and then the iteration finished accessing the Out array.

The optimized code is as follows:

1. for (int $kr = 0; kr < K; kr++$)
2. for (int $kc = 0; kc < K; kc++$)
3. for (int $r = 0; r < Rout; r++$)
4. for (int $c = 0; c < Cout; c++$)
5. $\#pragmaHLSPIPELINE$
6. for (int $cho = 0; cho < CHout; cho++$)
7. for (int $chi = 0; chi < CHin; chi++$)
8. $Out[cho][r][c] += In[chi][r + kr][c + kc] * W[cho][chi][kr][kc]$;

The Vivado HLS synthesis report is shown in Fig. 13, in which the value of the metric Initiation Interval is the ideal value of 1. In addition, the delayed clock cycles decreased by 36,503.

The latency and hardware resource usage values before and after the speedup are listed in Table II. It can be seen that the entire convolutional operation process was accelerated by a factor of 174.05, but the consumption of hardware resources also increased.

TABLE II
COMPARISON BEFORE AND AFTER ACCELERATION OF CONVOLUTION

	Before acceleration	After acceleration
Latency	1060833	6095
BRAM_18K	12%	12%
DSP48E	2%	36%
FF	-0%	9%
LUT	1%	23%

Acceleration on FPGA during pooling process

The pooling layers used in the entire CNN in this design were all maximum pooling, and the optimization process for the maximum pooling hardware parallel acceleration was similar to the convolutional operation, with the same optimization processes, including loop reconstruction, array division, loop expansion, and data access optimization. The entire pooling operation process was considerably accelerated, and the latency and hardware resource occupation values before and after the acceleration are listed in Table III. It can be seen that the entire convolutional operation process was accelerated by 61.35 times, but the occupied hardware resources also increased.

⊟ Summary

Latency		Interval		
min	max	min	max	Type
42598	42598	42598	42598	none

⊟ Detail

⊞ Instance

⊟ Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Kernel_Row_Kernel_Col	42596	42596	16	7	1	6084	yes

Fig. 10. Synthesis report of Vivado HLS after pipelining.

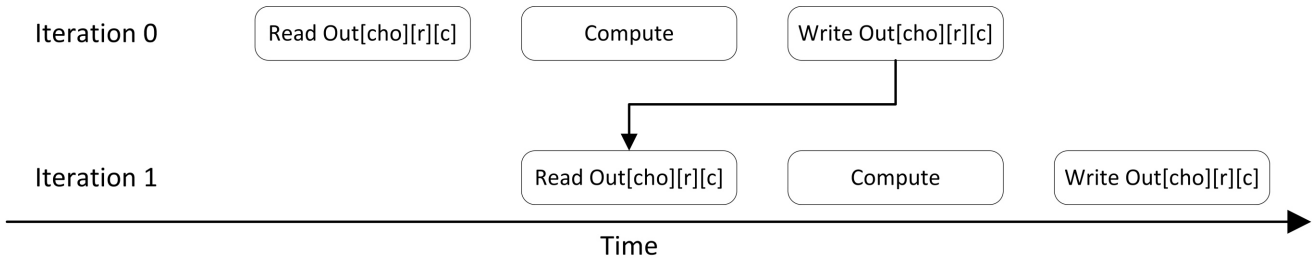

Fig. 11. Out array access flowchart.

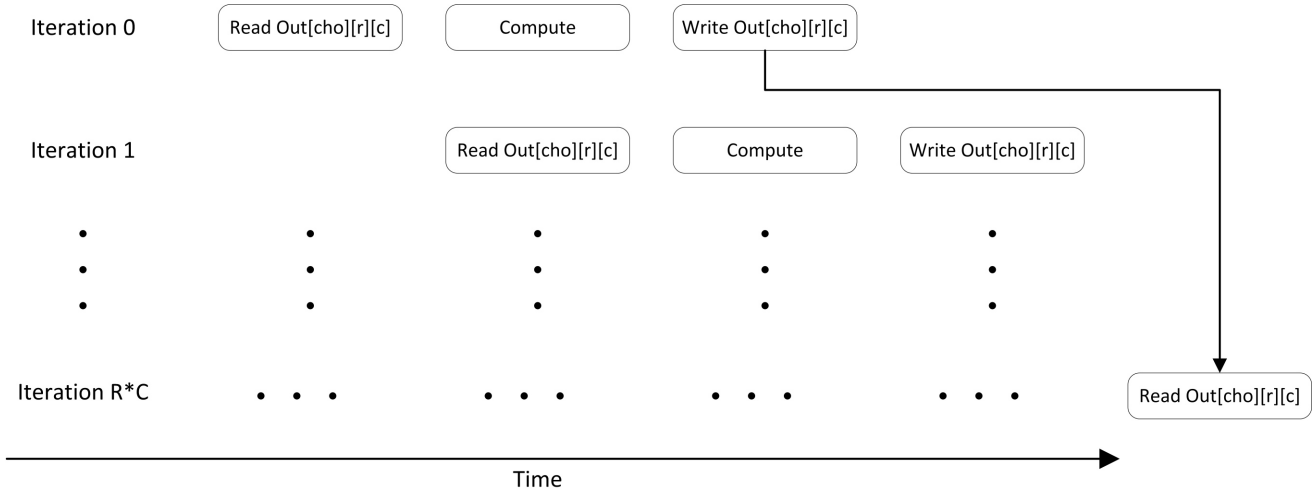

Fig. 12. Optimized Out array access flowchart.

⊟ Summary

Latency		Interval		
min	max	min	max	Type
6095	6095	6095	6095	none

⊟ Detail

⊞ Instance

⊟ Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Kernel_Row_Row_Col	6093	6093	11	1	1	6084	yes

Fig. 13. Vivado HLS report after pipelining.

TABLE III
COMPARISON BEFORE AND AFTER POOLING ACCELERATIO

	Before acceleration	After acceleration
Latency	60486	986
BRAM_18K	14%	22%
DSP48E	0%	0%
FF	-0%	2%
LUT	1%	9%

In this design, the C language code of the CNN algorithm was designed in Vivado HLS, and the corresponding optimization instructions were added. Test functions were written to perform software simulation and synthesis in C code, and it was finally package into the IP core. Fig. 14 shows the flowchart of the CNN implementation on the FPGA.

Fig. 15 shows the overall block design connection diagram of the system. The design included a CNN acceleration IP core generated using Vivado HLS, input and output IP cores for video streaming, temperature measurement IP cores, and some other peripheral drivers to meet the overall requirements of the system.

IV. PS END DEVELOPMENT BASED ON PYNQ FRAMEWORK

A. Introduction to PYNQ

PYNQ is an open source project from Xilinx that uses the Python language for FPGA development. The PYNQ framework is structured in three layers: hardware, which focuses on FPGA design; software, which loads the Linux kernel with Python; and applications, which are based on Jupyter Notebook. The PYNQ framework provides complete PS access to the Python library of PL resources. This allows upper-layer applications to obtain implementation details of the underlying hardware, which efficiently and quickly creates high-performance applications. The design was based on PYNQ to implement the face detection, HTTP server construction and implementation, IP core invocation on the PL end, and implementation of the overall logical functionality of the system.

B. OpenCV-based Face Detection

OpenCV is an open source cross-platform computer vision and machine learning software library. It provides interfaces to C, C++, Python, and other programming languages, providing a variety of algorithms for computer vision and image processing, which can help developers to quickly and efficiently complete the development of image processing and other applications. OpenCV encapsulates the Haar feature-based Adaboost cascade classifier [21], which allows fast human feature detection. In the current design, OpenCV was used in the PYNQ framework for face detection. The cascade classifier haarcascade frontalface default.xml was used to implement the face detection. The CascadeClassifier() function was called to load the cascade classifier, the detectMultiScale() function was used to complete the face detection, and the face coordinates, height, and width were returned. The key code is as follows:

1. import cv2 2. Load cascade classifier file 3. faceCascade = cv2.CascadeClassifier(’. /haarcascade_ frontalface_default.xml’) 4. Read in images in grayscale 5. img = cv2.imread(’. /img.jpg’, cv2.IMREAD_GRAYSC ALE) 6. Call the detection function and return the coordinates of the face and its height and width 7. faces = faceCascade.detectMultiScale(img)

C. HTTP Server Construction and Implementation

The design imported the socket network programming module in the PYNQ framework and implemented HTTP server construction based on the transmission control protocol. Based on the HTTP protocol request-response model, the client sent a request message by GET. The server provided the corresponding services to each client requesting connection. This included receiving the request message, parsing the message, determining whether the requested data were in the service range, and responding to the message. The process flow of this service sub-program is shown in Fig. 16.

D. Logic for Implementing System Functions on PYNQ

The design of the overall logic function of the system was implemented at the PS end. Fig. 17 shows the process flowchart of the image processing part, which realized real-time face image acquisition, in which the OpenCV and CNN IP cores were called for mask wearing detection. Figs. 18 and 19 show the flowcharts of the IR distance detection module and IR temperature measurement driver module, respectively, which automatically began temperature conversion and compensation when a human body approached the measurement area.

V. WEB PAGE DESIGN AND DEVELOPMENT

The web page design of the contactless intelligent epidemic prevention system mainly provided an updated display of real-time temperature data, along with the ability to save historical temperature data and view the recognition result of the latest target detection. The CSS language was used to configure the overall layout format of the page, and the JavaScript scripting language was used to process some simple animation effects and the real-time sending of a request to the service for the dynamic updating of temperature data. Previews of the system history data and target detection results were also available. The web home page, history data, and target detection page previews are shown in Figs. 20, 21, and 22, respectively.

VI. SYSTEM EXPERIMENT RESULTS AND ANALYSIS

A. Experimental Environment

The implementation environment of this design required a combination of hardware, including a PC host and FPGA development board, and software, including development tools such as deep learning libraries and FPGA design software, as listed in Table IV.

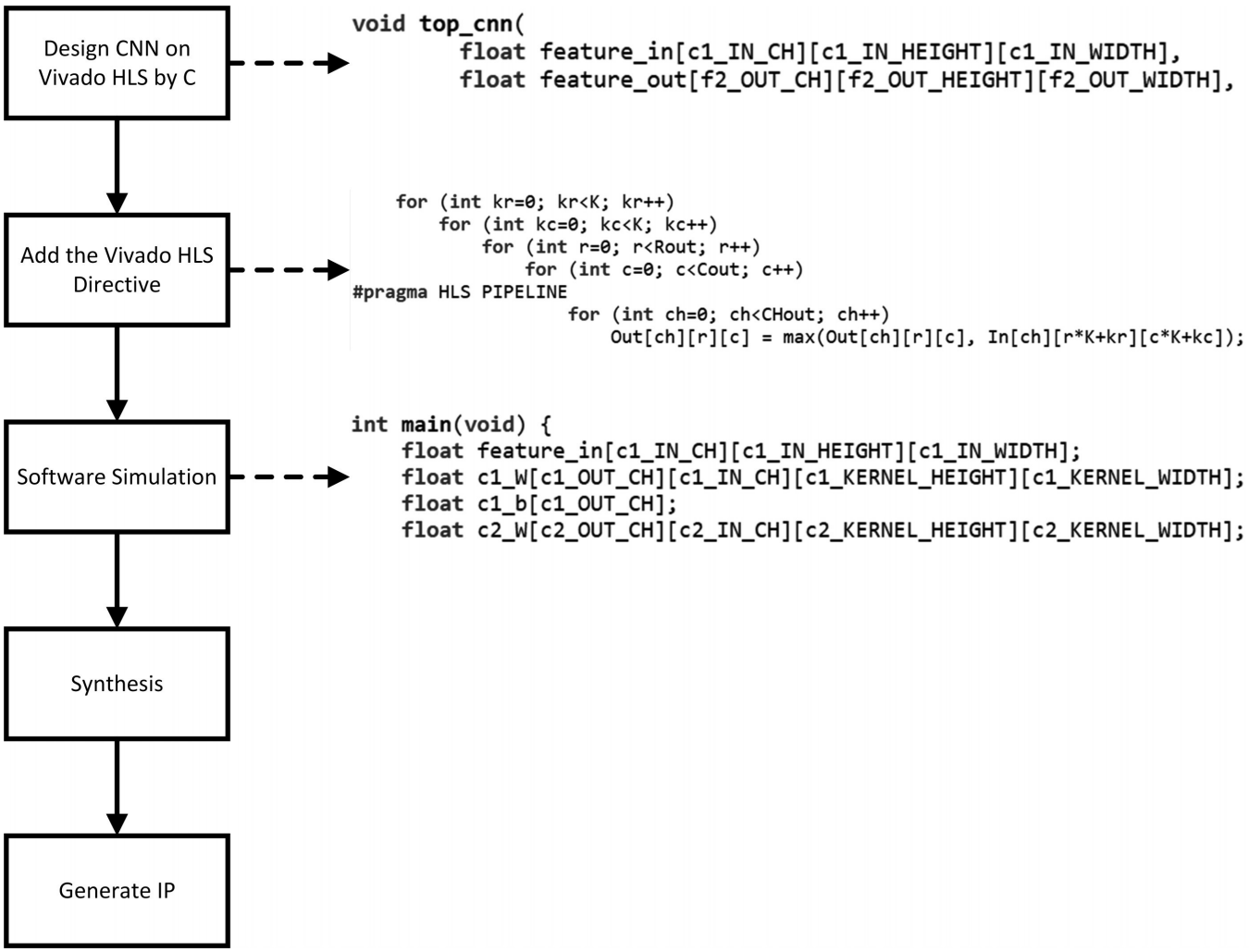

Fig. 14. Flowchart of CNN implementation on FPGA.

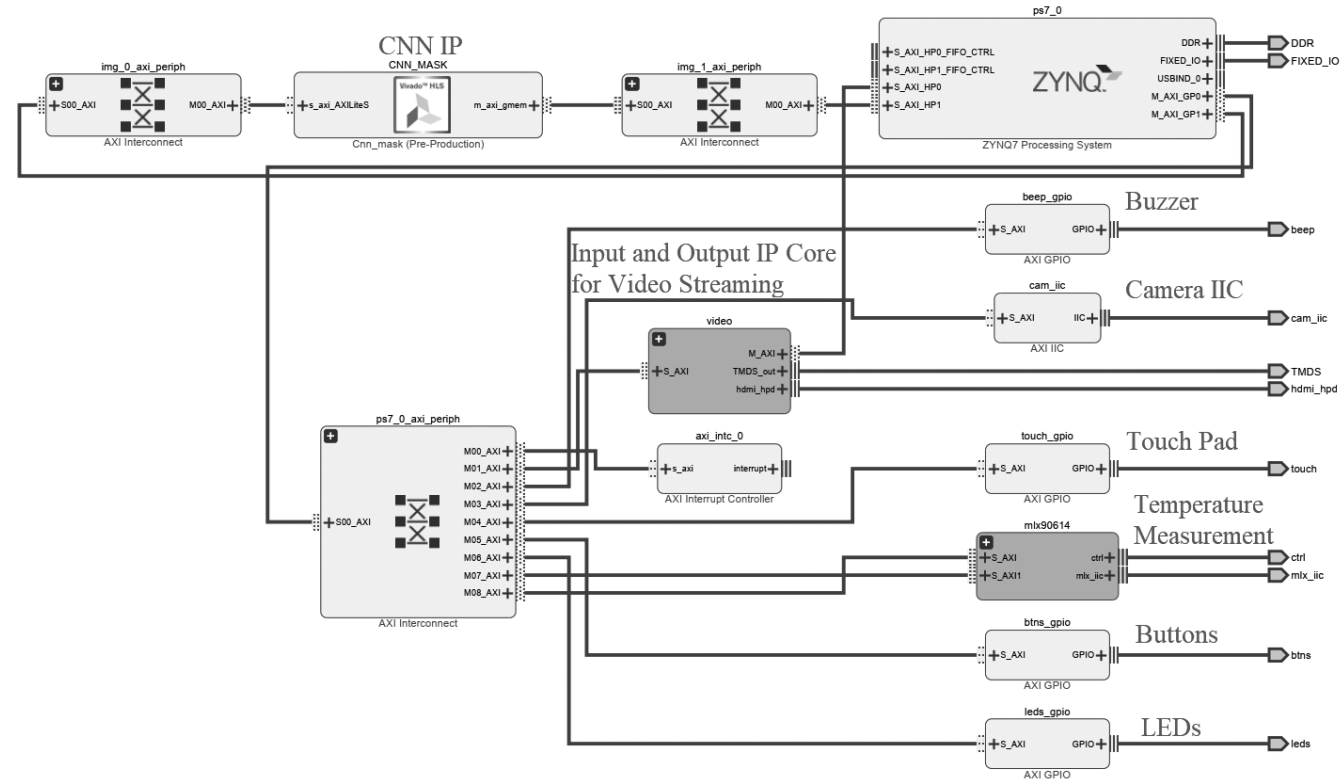

Fig. 15. Overall block design connection diagram.

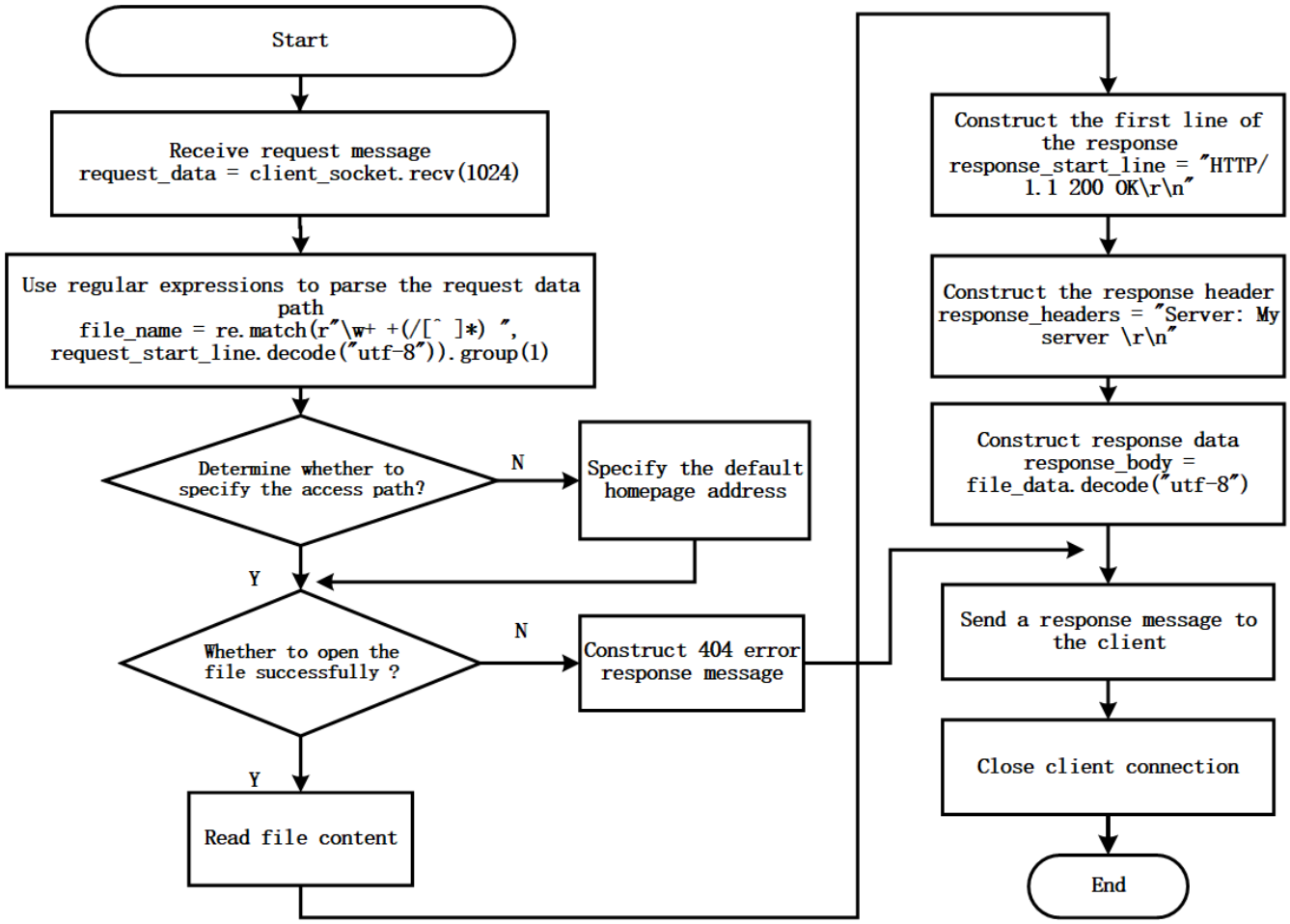

Fig. 16. Flowchart of HTTP service sub-program.

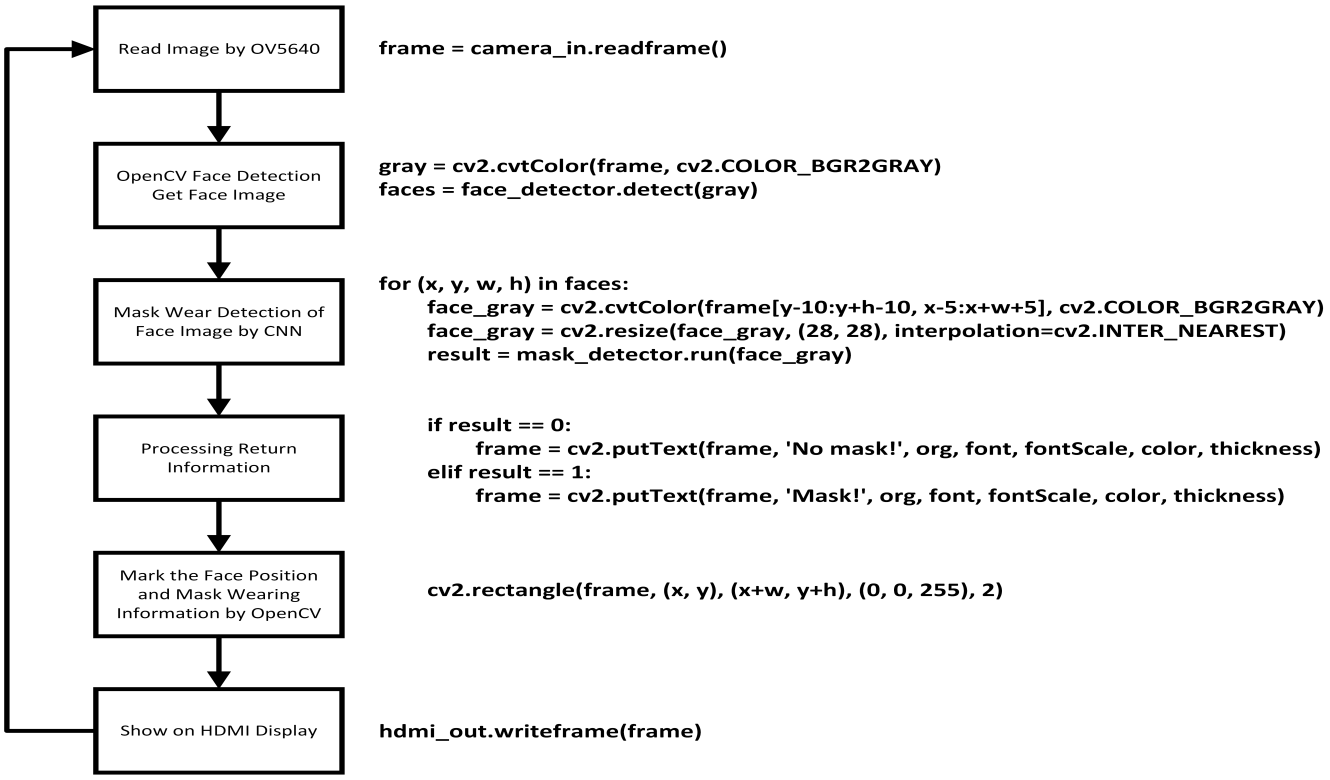

Fig. 17. Flowchart of the image processing program.

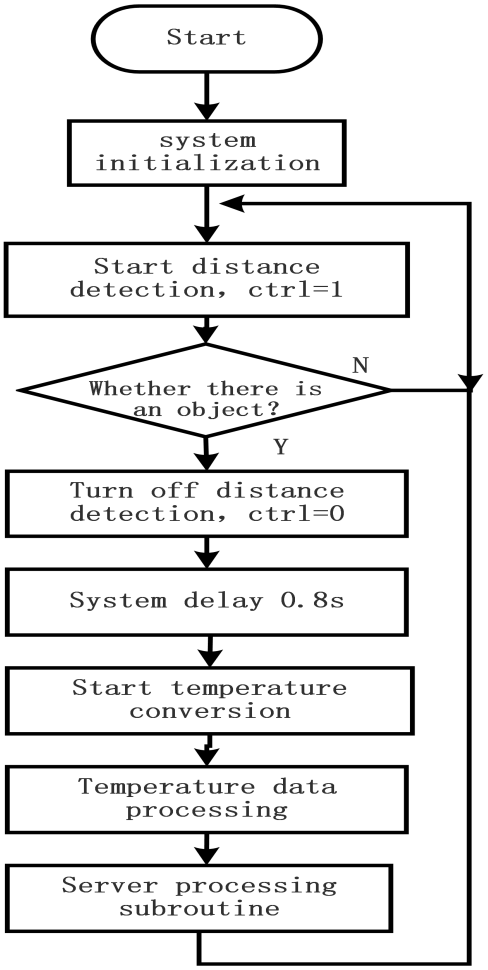

Fig. 18. Flowchart of the IR distance detection driver.

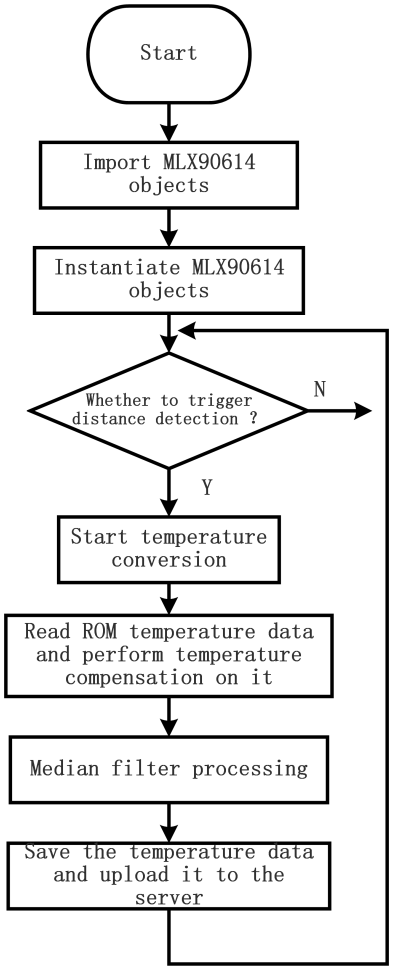

Fig. 19. Flowchart of the IR temperature measurement driver module.

TABLE IV
HARDWARE AND SOFTWARE REQUIRED FOR IMPLEMENTATION

Name	Model, Version
PC host CPU	Intel(R) Xeon(R) Gold 6226 CPU 2.70 GHz
PC host memory	8G
FPGA model	xc7z020clg400-2
PC host operating system	Ubuntu 18.04.1
Deep learning library	TensorFlow 1.15
Image processing software	OpenCV 4.4.0
Advanced integrated software	Vivado HLS 2018.3
FPGA set design software	Vivado 2018.3

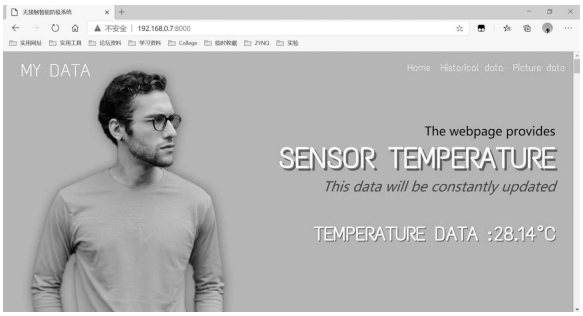

Fig. 20. Preview of web home page.

B. Experimental Test Protocol

The system needed to perform calibration and compensation for the collected temperature data to prevent errors in the measurement accuracy caused by environmental and other factors. Therefore, the data from experimental tests were particularly important. The test scheme used in the experiment was a water bath to provide a constant temperature reference source, where the value of a water thermometer was read for a temperature comparison, and the MLX90614 temperature acquisition module was used to collect temperature data at different distances. The measured temperature data were curve fitted to the measured distances under different conditions, as shown in Fig. 23.

An analysis of the data and the fitted curve showed that the temperature collected by the sensor increased with an increase in the target temperature and decreased with an increase in the measurement distance. When the measurement distance was approximately 6 cm, the error range was within $\pm$ 0.3 °C. Therefore, a distance of 6 cm was chosen as the optimal measurement distance.

C. Human Temperature Compensation Algorithm and Analysis of its Measurement Data

When a sensor measures temperature, it first takes a predictive measurement of the ambient temperature (sensor temperature), and then calculates the temperature to be measured [24]. Thus, the human body temperature compensation was an estimate of the actual human body temperature based

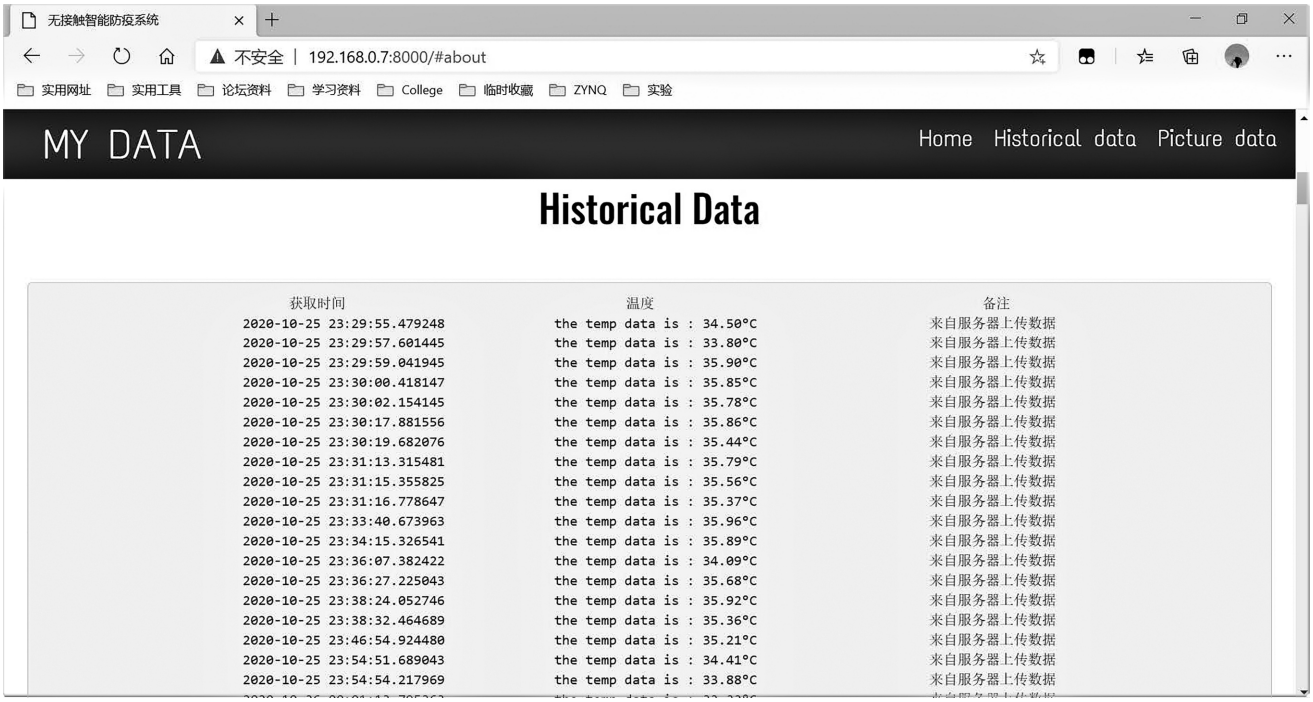

Fig. 21. Preview of historical data page.

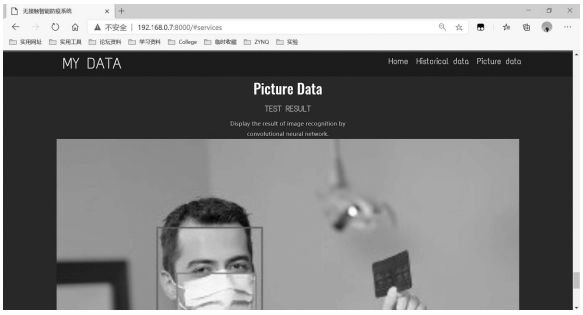

Fig. 22. Preview of target detection results.

on the object temperature and ambient temperature using the following equation:

$$T_{body} = T_o + \alpha \times (T_o - T_a) + bias \tag{3}$$

where T_{body} is the human body temperature, T_o is the object temperature, α is the compensation coefficient, T_a is the ambient temperature, and bias is the temperature compensation. The T_a value used for the experimental data test condition was above 25 °C, and the compensation coefficient and bias were calculated in four temperature ranges for T_o, namely 25.0-32.0 °C, 32.0-34.5 °C, 34.5-36.5 °C, and above 36.5 °C. The experiment results are shown in Fig. 24.

After utilizing the compensation algorithm for the human body temperature, the temperature acquisition system performed the following experimental data measurements by comparing the results obtained with a forehead thermometer. The results are listed in Table V. It was found that the measurement error was ± 0.3 °C after several measurements.

TABLE V
BODY TEMPERATURE DATA COLLECTION

Number of measurements	Measurement data (°C)	Forehead thermometer (°C)	Error (%)
1	36.66	36.5	0.44
2	36.86	36.8	0.16
3	36.78	36.7	0.22
4	36.48	36.3	0.50
5	36.63	36.5	0.36
6	36.39	36.3	0.25
7	36.29	36.1	0.53
8	36.87	36.8	0.19
9	36.52	36.5	0.05
10	36.75	36.6	0.41
11	36.77	36.5	0.74
12	36.53	36.3	0.63
13	36.86	36.8	0.16
14	36.49	36.3	0.52
15	36.95	36.8	0.41
16	36.58	36.4	0.49
17	36.67	36.6	0.19
18	36.86	36.8	0.16
19	36.69	36.6	0.25
20	36.89	36.7	0.52

D. Testing of Multi-device Access to HTTP Server

After the mapping of the LAN ports of the contactless intelligent epidemic prevention system was completed, multi-device access to the target web page of the HTTP server on ZYNQ was performed to test laptop, mobile phone, and iPad access to the web page [25]. The test results are shown in Figs. 25 and 26.

VII. CONCLUSION AND RECOMMENDATION

This paper described the design process for a contactless intelligent epidemic prevention system based on ZYNQ. The main efforts included the following: (1) When a temper-

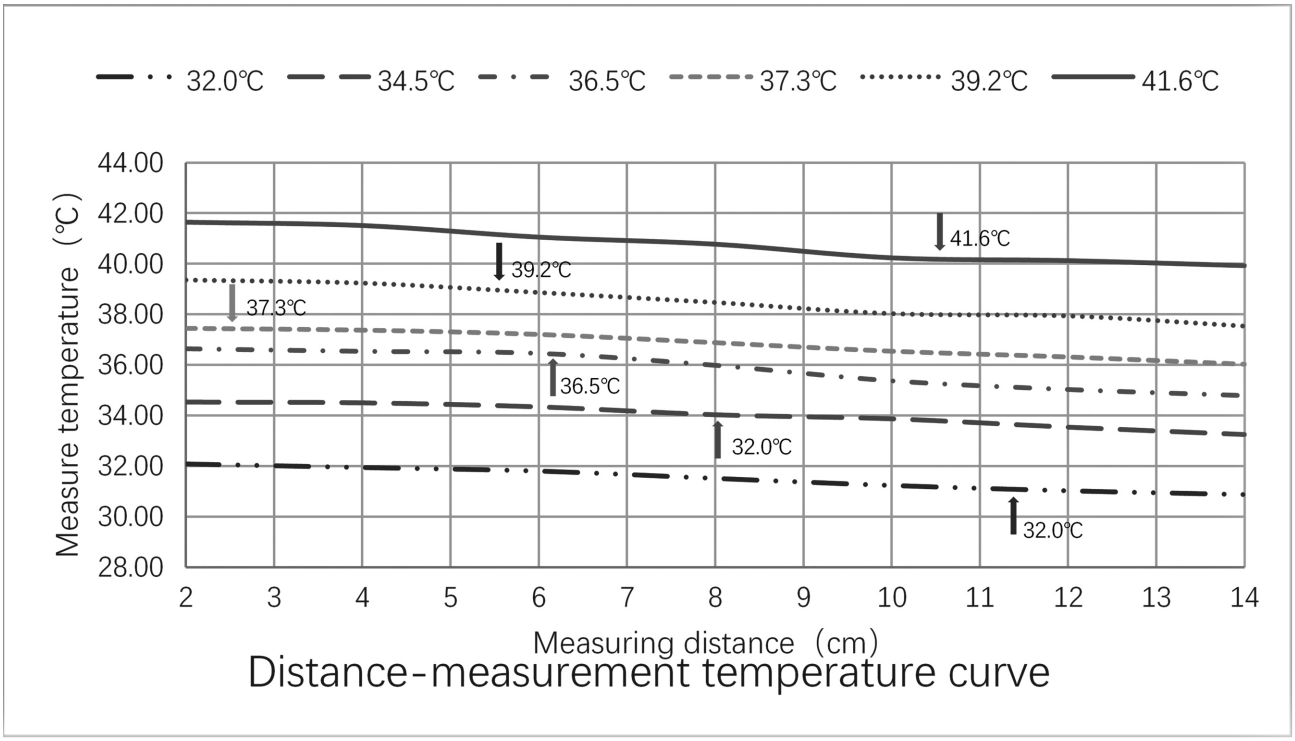

Fig. 23. Distance vs. measured temperature curve.

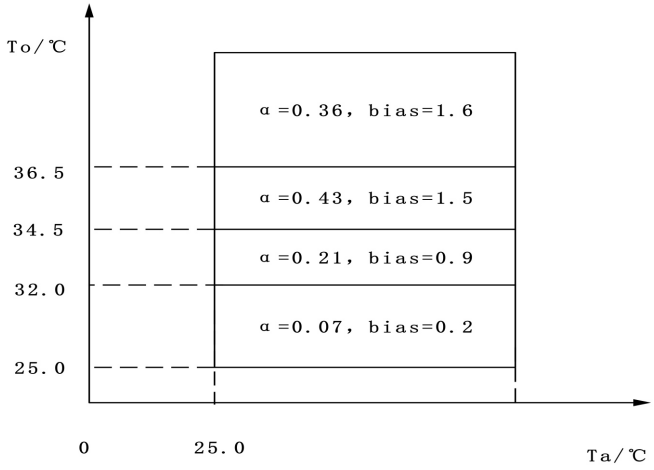

Fig. 24. Compensation factor and bias.

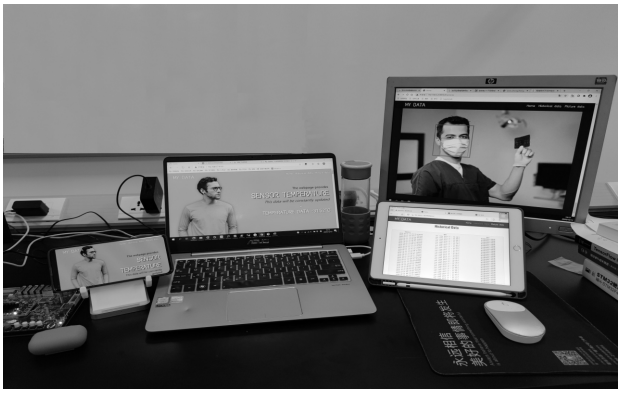

Fig. 25. Access tests by mobile phone, PC, iPad and other devices.

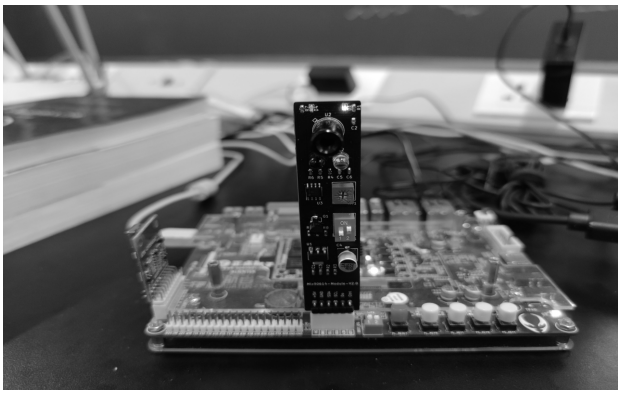

Fig. 26. ZYNQ operation in real time.

ature compensation algorithm was used, the human body temperature measurement error was within $\pm$ 0.3 °C, and the automatic non-contact measurement of a persons temperature was realized. (2) An OV5640 camera was used in real time to obtain a target image, which was stored and accessed through a high-speed bus and external memory. (3) Xilinxs open-source framework PYNQ was used for software development, to control the ZYNQ with Python programs, and to call OpenCV to complete face detection. (4) The advanced synthesis tool Vivado HLS was utilized to complete the development of the FPGA program, and the CNN was deployed on the FPGA to complete the mask recognition. By calculating, dividing, expanding, and streamlining the calculation process, a hardware parallel acceleration process for the CNN was realized. (5) An HTTP server was established, and devices on a LAN could access this server to fetch historical temperature data. The system still needs further improvement

in some aspects. For example, the reference data used by the fitting temperature compensation algorithm came from a water bath rather than a standard blackbody calibration source. The service scope provided by the server was only in the LAN, and no identity authentication mechanism was added to limit access to the page, which lacked security. If the above issues could be resolved, the contactless intelligent epidemic prevention system would be more suitable for the needs of an actual application.

REFERENCES

[1] X. Liu and S. Zhang, "Covid: Face masks and humanto-human transmission," *Influenza and Other Respiratory Viruses*, vol. 14, no. 4, p. 472, 2020.

[2] S. Hanaei and N. Rezaei, "Covid-19: developing from an outbreak to a pandemic," *Archives of medical research*, vol. 51, no. 6, p. 582, 2020.

[3] K. Nikolopoulos, S. Punia, A. Schäfers, C. Tsinopoulos, and C. Vasilakis, "Forecasting and planning during a pandemic: Covid-19 growth rates, supply chain disruptions, and governmental decisions," *European journal of operational research*, vol. 290, no. 1, pp. 99–115, 2021.

[4] S. Feng, C. Shen, N. Xia, W. Song, M. Fan, and B. J. Cowling, "Rational use of face masks in the covid-19 pandemic," *The Lancet Respiratory Medicine*, vol. 8, no. 5, pp. 434–436, 2020.

[5] M. Polsinelli, L. Cinque, and G. Placidi, "A light cnn for detecting covid-19 from ct scans of the chest," *Pattern recognition letters*, vol. 140, pp. 95–100, 2020.

[6] R. Padilla, S. L. Netto, and E. A. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. Niteroi, Brazil: IEEE, 2020, pp. 237–242.

[7] M. S. Ejaz, M. R. Islam, M. Sifatullah, and A. Sarker, "Implementation of principal component analysis on masked and non-masked face recognition," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*. Dhaka, Bangladesh: IEEE, 2019, pp. 1–5.

[8] X. Wu, D. Sahoo, and S. C. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[9] N. A. Abdullah, M. J. Saidi, N. H. A. Rahman, C. C. Wen, and I. R. A. Hamid, "Face recognition for criminal identification: An implementation of principal component analysis for face recognition," in *AIP Conference Proceedings*, vol. 1891, no. 1. Miri, Malaysia: AIP Publishing LLC, 2017, p. 020002.

[10] M. Yang, J. Crenshaw, B. Augustine, R. Mareachen, and Y. Wu, "Adaboost-based face detection for embedded systems," vol. 114, no. 11, 2010, pp. 1116–1125.

[11] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, "Fighting against covid-19: A novel deep learning model based on yolo-v2 with resnet-50 for medical face mask detection," *Sustainable Cities and Society*, vol. 65, p. 102600, 2021.

[12] M. Loey and G. Manogaran, "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic," *Measurement*, vol. 167, p. 108288, 2021.

[13] S. Ghaffari and S. Sharifian, "Fpga-based convolutional neural network accelerator design using high level synthesize," in *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*. Tehran, Iran: IEEE, 2016, pp. 1–6.

[14] M. Shivaraju and S. Krishnappa, "A new parallel dsp hardware compatible algorithm for noise reduction and contrast enhancement in video sequence using zynq-7020," *International Journal of Computer Aided Engineering and Technology*, vol. 13, no. 1-2, pp. 14–27, 2020.

[15] J. P. Smith, J. Bailey, J. Tuthill, L. Stefanazzi, G. Cancelo, K. Treptow, and B. A. Mazin, "A high-throughput oversampled polyphase filter bank using vivado hls and pynq on a rfsoc," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 241–252, 2021.

[16] H. Irmak, F. Corradi, P. Detterer, N. Alachiotis, and D. Ziener, "A dynamic reconfigurable architecture for hybrid spiking and convolutional fpga-based neural network designs," *Journal of Low Power Electronics and Applications*, vol. 11, no. 3, p. 32, 2021.

[17] A. Sudianto, Z. Jamaludin, A. A. A. Rahman, S. Novianto, and F. Muharrom, "Automatic temperature measurement and monitoring system for milling process of aa6041 aluminum aloy using mlx90614 infrared thermometer sensor with arduino," *Journal of Advanced Research in Fluid Mechanics and Thermal Sciences*, vol. 82, no. 2, pp. 1–14, 2021.

[18] M. B. Chaniago, L. G. M. Farizt, K. A. Putra, I. Mubarok, and R. R. O. Mahanandi, "Design and development of smart noncontact thermometer using the mlx90614 sensor and microcontroller-based face recognition," *Solid State Technology*, vol. 63, no. 4, pp. 4871–4878, 2020.

[19] B. P. Senyurek V Y, Imtiaz M H, "A cnn-lstm neural network for recognition of puffing in smoking episodes using wearable sensors," *Biomedical Engineering Letters*, vol. 10, no. 2, pp. 195–203, 2020.

[20] F. Kästner, B. Janßen, F. Kautz, M. Hübner, and G. Corradi, "Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Vancouver, BC, Canada: IEEE, 2018, pp. 154–161.

[21] Y.-H. Chen, C.-P. Fan, and R. C.-H. Chang, "Prototype of low complexity cnn hardware accelerator with fpga-based pynq platform for dual-mode biometrics recognition," in *2020 International SoC Design Conference (ISOCC)*. Yeosu, Korea (South): IEEE, 2020, pp. 189–190.

[22] L. Stornaiuolo, M. Santambrogio, and D. Sciuto, "On how to efficiently implement deep learning algorithms on pynq platform," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Hong Kong, China: IEEE, 2018, pp. 587–590.

[23] L. Huang, J. Guo, and Z. Gao, "A face recognition system on embedded device," *Journal of Computers*, vol. 31, no. 1, pp. 176–183, 2020.

[24] J. Chen, J.-P. Wang, T.-Y. Shen, D.-X. Xiong, and L.-Q. Guo, "High precision infrared temperature measurement system based on distance compensation," in *ITM Web of Conferences*, vol. 12. GuanDong, China: EDP Sciences, 2017, p. 03021.

[25] J. Li, W. Huang, Z. Dai, and X. Bian, "Implementation of biometrics recognition system based on zynq soc platform and cloud server," in *Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City*, Shanghai China, 2019, pp. 218–222.

LINGZHI WANG was born in Fujian, China in December 1981. She is an Associate Professor in Department of Electronic and Information Engineering, Xiamen City University,China. She received her Masters of Engineering in Microelectronics and Solid State Electronics from Fuzhou University in 2006. Her fields of expertise include the Internet of Things, image processing and embedded systems.

JIANYUAN WANG was born in Fujian, China, in 1998, received the B.E degree from Minnan Normal University, China, in 2021. He mainly engaged in electronic system design research.

YINGJING XU was born in Fujian, China, in 1999, received the B.E degree from Minnan Normal University, China, in 2021. He mainly engaged in embedded systems and machine learning.