


Article

Intelligent Security Monitoring System Based on RISC-V SoC

Wenjuan Wu , Dongchu Su, Bo Yuan and Yong Li *

College of Computer Science, National University of Defense Technology, Changsha 410073, China; wuwenjuan@nudt.edu.cn (W.W.); sudongchu@nudt.edu.cn (D.S.); yuanbo18@nudt.edu.cn (B.Y.)

* Correspondence: yongli@nudt.edu.cn; Tel.: +86-13723890445

Abstract: With the development of the economy and society, the demand for social security and stability increases. However, traditional security systems rely too much on human resources and are affected by uncontrollable community security factors. An intelligent security monitoring system can overcome the limitations of traditional systems and save human resources, contributing to public security. To build this system, a RISC-V SoC is first designed in this paper and implemented on the Nexys-Video Artix-7 FPGA. Then, the Linux operating system is transplanted and successfully run. Meanwhile, the driver of related hardware devices is designed independently. After that, three OpenCV-based object detection models including YOLO (You Only Look Once), Haar (Haar-like features), and LBP (Local Binary Pattern) are compared, and the LBP model is chosen to design applications. Finally, the processing speed of 1.25 s per frame is realized to detect and track moving objects. To sum up, we build an intelligent security monitoring system with real-time detection, tracking, and identification functions through hardware and software collaborative design. This paper also proposes a video downsampling technique. Based on this technique, the BRAM resource usage on the hardware side is reduced by 50% and the amount of pixel data that needs to be processed on the software side is reduced by 75%. A video downsampling technology is also proposed in this paper to achieve better video display effects under limited hardware resources. It provides conditions for future function expansion and improves the models' processing speed. Additionally, it reduces the run time of the application and improves the system performance.

Keywords: FPGA; embedded system; object tracking; object detection; OpenCV



Citation: Wu, W.; Su, D.; Yuan, B.; Li, Y. Intelligent Security Monitoring System Based on RISC-V SoC. *Electronics* **2021**, *10*, 1366. <https://doi.org/10.3390/electronics10111366>

Academic Editor: George A. Papakostas

Received: 7 May 2021

Accepted: 3 June 2021

Published: 7 June 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the advent of the era of artificial intelligence, the applications of intelligent security systems [1–3] become more and more diverse. As one of the branches, the intelligent security monitoring system [4,5] plays an increasingly important role in social security. The traditional security monitoring system is mainly based on a computer and needs human assistance to realize video monitoring, inspection, recording, and analysis, thus requiring a large workforce and increased material costs. The intelligent security monitoring system studied in this paper is based on an embedded platform and can be deployed on mobile terminals. Based on the traditional security monitoring system, computer vision technology [6–8] is employed to perceive external information. Especially, the object detection model [9] is exploited to record and analyze the video information, overcoming the limitation of the traditional security monitoring system with limited human resources. RISC-V is the fifth-generation simplified instruction set architecture developed by the University of California, Berkeley [10–12]. As a simple, free, and open architecture, it allows anyone to design, manufacture, and sell RISC-V chips or software. RISC-V SoC realizes IP (Intellectual Property) sharing in the real sense and promotes the establishment and development of hardware open-source technology platforms and shared open-source ecosystems. Currently, many relevant research results based on RISC-V SoC have emerged. Farshchi et al. [13] integrated NVDLA into a RISC-V SoC on the Amazon Cloud FPGA using

Firesim (Precise Cycle FPGA Accelerator). Flamand et al. [14] proposed a multi-GOPS RISC-V SoC-based fully programmable edge computing engine to implement advanced sensor algorithms based on machine learning. Zhong et al. [15] presented a RISC-V SoC based on Visible Light Communication (VLC) for mobile payment applications. Bailey et al. [16] demonstrated a signal analysis SoC consisting of a general-purpose RISC-V core with vector extensions and a fixed-function signal-processing accelerator. Enrique et al. [17] used open-source software to develop AI IoT applications on RISC-V SoC. The advantages of the RISC-V architecture in security and power consumption promote the application of RISC-V in the Internet of Things (IoT), mobile terminal, edge computing, and other fields. However, the application of RISC-V instruction set architecture is missing in intelligent security, and the combination of hardware and software is still lacking in related studies. In order to use the advantages of RISC-V in power consumption and security and to enrich the ecological construction of RISC-V, this paper proposes an intelligent security monitoring system based on RISC-V SoC to broaden the application of RISC-V in the embedded domain. This paper constructs an intelligent security monitoring system based on RISC-V SoC, and the system consists of hardware parts and software parts. The hardware part includes RISC-V SoC and peripherals, and the software part includes the Linux operating system and application programs. Under the condition of limited hardware resources, this paper first optimizes the hardware control logic design of the RISC-V SoC according to the functional requirements of the system and then compiles, transplants, and runs the Linux operating system to control the necessary peripherals. In the design of application programs, the LBP model is used for data processing, and then, the PID (Proportional-Integral-Differential) algorithm is used to control the servo according to the processing results. Finally, the moving object within the monitoring area can be detected and tracked, and the recorded video is uploaded through the network port. The server performs provides data storage and sharing, and subsequent face recognition and identity recognition can be realized by comparing the face data in the database.

2. Overall System Design

2.1. System Composition

According to the functional requirements of the system, the detection and tracking of moving objects need to be implemented on a RISC-V SoC with peripheral controllers such as the camera, HDMI, and servo. Additionally, the Linux operating system and running drivers as well as applications in SoC need to be transplanted. The application performance mainly depends on the selected model. The overall structure of the system is illustrated in Figure 1.

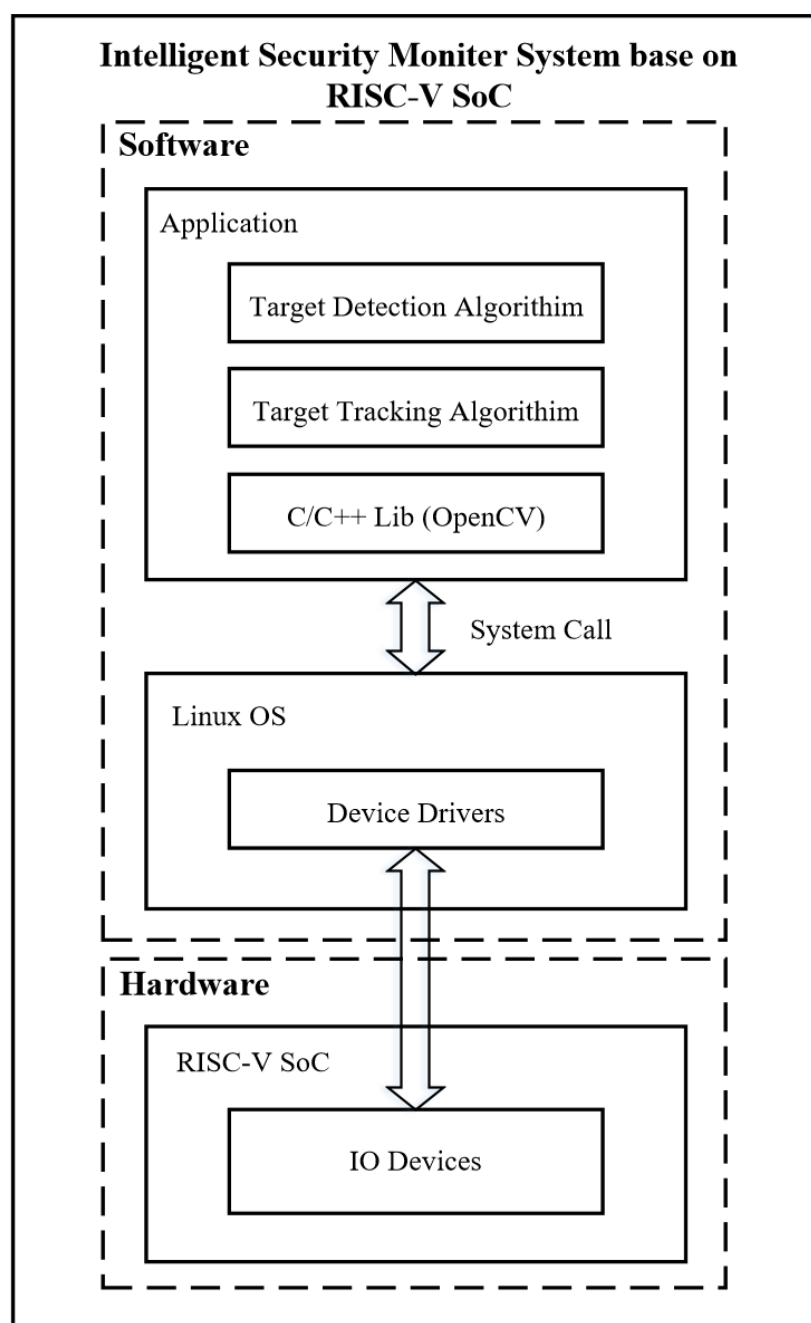


Figure 1. Overall system organization.

2.2. Introduction of the Platform and Design Tools

The hardware platform used in this system is the Nexys-Video Artix-7 FPGA embedded development platform manufactured by Digilent company. Among Digilent's development board series, the Nexys-Video is an efficient tool specially designed for audio/video applications. It is equipped with a high-bandwidth external memory, three high-speed digital video ports, and a 24-bit audio codec design and has standard communications, users, and expansion peripherals. Specifically, the communication peripherals include ethernet, USB-UART, and high-speed USB interfaces. The on-board user peripherals include switches, buttons, led, and OLED displays. The expansion peripherals include one FMC connector and four Pmod ports to allow additional peripherals to be added. The integrated design tool used in this system is the Vivado design suite. The project was established to complete the addition of IP and HDL (Hardware Description Language) design. The hardware circuit that meets the functional requirements of the system is real-

ized through the interconnection between IPs and RTL (Resistor Transistor Logic) modules. After the functional simulation, synthesis, implementation, and bitstream generation, the generated bitstream is programmed to the FPGA of the development board, and the system hardware platform is constructed.

3. System Hardware Design and Implementation

The key component of the hardware implementation of the system mainly lies in RISC-V SoC and its peripherals. As shown in Figure 2, the FPGA of the Nexys Video Artix-7 development platform is used to implement the RISC-V SoC and, together with the hardware resources on the development board, to form a logic circuit that can realize specific functions.

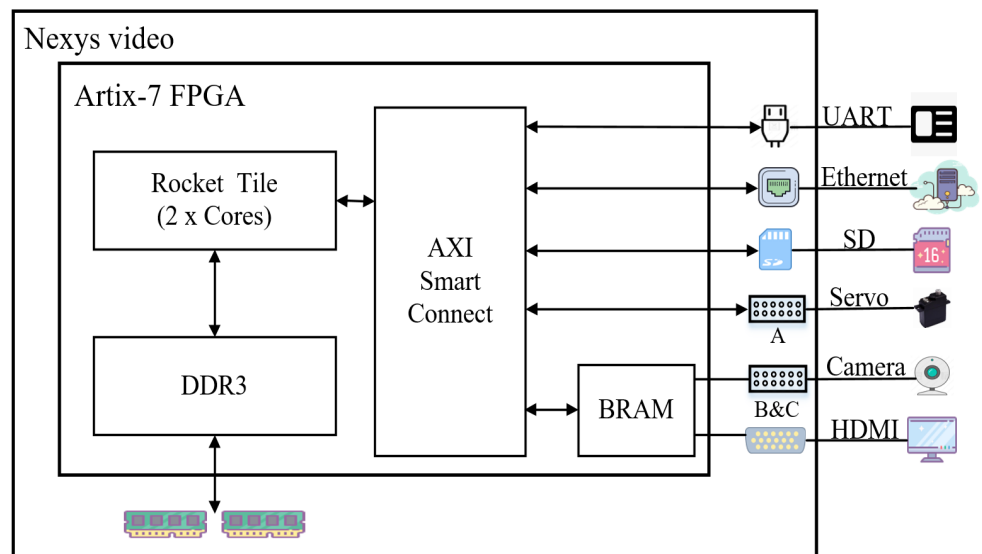


Figure 2. System hardware structure.

RISC-V SoC is mainly composed of a processor core, DDR3, and input and output modules. These modules are interconnected and communicated using the AXI bus. Meanwhile, each submodule in the modules is connected through the crossbar AXI smart connect so that the processor can realize the read and write operations of the peripheral data in a one-master-multiple-slave manner. In SoC, the AXI bus is responsible for the communication between modules, such as sending and receiving addresses and data. After the master and slave devices are connected, any read or write operation has an address that specifies the operation object. The final design needs to be validated to ensure that the design is correct. The interconnection between the modules is shown in Figure 3.

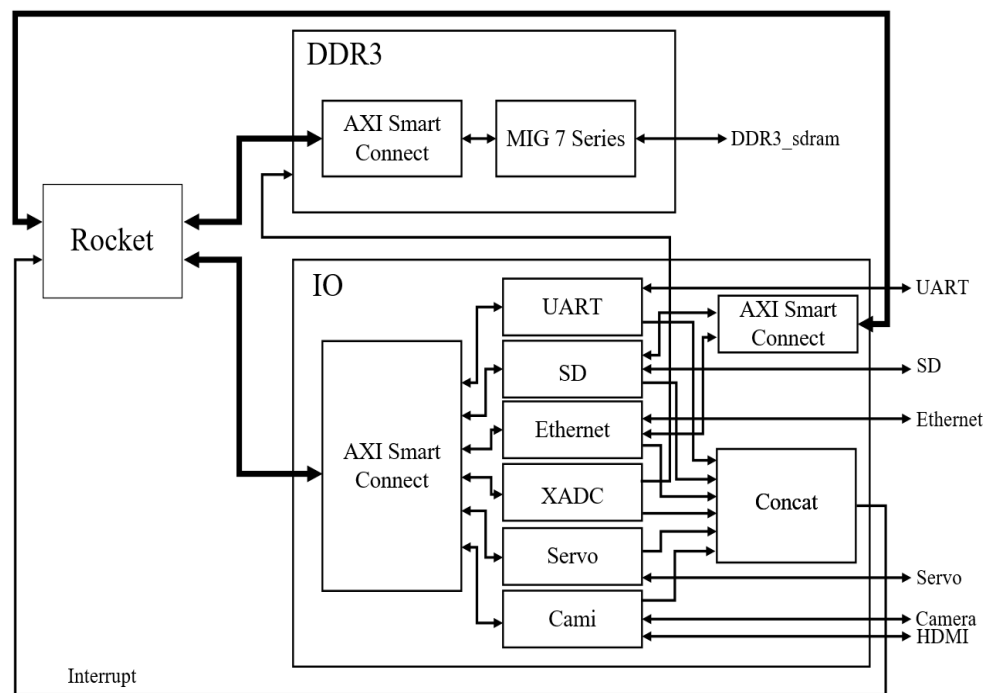


Figure 3. Module interconnection.

The module interconnection diagram clearly shows the connection between the modules. The following describes the design and implementation of each module in detail. There have been many open-source RISC-V processor core designs, such as the Boom developed by the University of Berkeley, the Ariane developed by the University of Zurich, and the Shakti series developed by the Indian Institute of Technology. Some of the designs are successful in streamer [18–21], which indicates that RISC-V has a broad application prospect. In our system, the processor core is a 64-bit RISC-V Rocket softcore designed by the University of Berkeley [22]. The core exploits the classic five-stage pipeline, sequential execution, single launch, and it supports various branch predictions. As a classic and configurable processor core of RISC-V series, the Rocket core can fully support the operating system's transplantation and operation. The paper configures the processor to have dual cores as needed. As part of the on-board resources, the DDR3 can be regarded as the SoC memory for storing some of the data and instructions required by the processor core. The DDR3 MIG (Memory Interface Generator) designed in SoC is a converter between the DDR3 interface and the user logic control interface. The parameters of the MIG need to be configured according to actual conditions, including system clock and reference clock, data width, pin assignments, etc. The processor core performs read and write operations of DDR3 through the DDR3 MIG. The input and output modules mainly include the AXI smart connect, UART serial port module, ethernet module, SD card controller, servo module, OV5640 camera module, and HDMI display module. The AXI smart connect can add peripherals with the AXI interface as needed, which makes the hardware design scalable. Additionally, it helps the processor control multiple inputs and output devices while synchronizing signals and reset signals from different clock domains to ensure that device data read and write can meet the timing requirements. After the port signals of the remaining modules are assigned and mapped, the corresponding peripherals can be controlled. The servo module, OV5640 camera module, and HDMI display module are significant for implementing the system functions. Therefore, this paper conducts an independent HDL language design to implement hardware control logic for the above three modules. Based on this, the RTL function test stage and the oscilloscope signal debugging stage are refined to ensure the best system function realization with the lowest hardware resource consumption. Compared with the design of the servo module, the control logic that needs to be implemented for the OV5640 camera module and HDMI display module

is relatively complicated. Thus, it is described in detail in the following part. The system function aims to display the output pixel data of the HDMI device. The camera pixel clock and the HDMI output clock differ in frequency, which makes it necessary to provide a read–write buffer of a specific pixel data capacity. Although the DDR3 has outstanding advantages as a large-capacity high-speed storage device, more control logic needs to be adjusted and the subsequent driver programming is complicated. The BRAM (Block Random Access Memory) IP provided by Vivado can be used as a buffer. To ensure the correct output of the display device, the BRAM should be configured to buffer one data frame. The camera captures a frame with a resolution of 1024×768 , and the pixel in the RGB565 format has a total of 16 bits of data. According to this, the BRAM size is at least 12 Mb, which accounts for more than 99% of the board-level storage resources. In this case, the design fails in synthesis. Meanwhile, to improve the data transmission efficiency and system performance, it is also necessary to design a BRAM dedicated to data processing for the system, which is the same as the BRAM required for output display. In this case, the demand for storage resources is much larger than the actual storage capacity of the development board. To solve this problem, a video downsampling technology is adopted by the design, and two identical BRAM IPs are successfully designed. Realizing the typical display and transmission of image data through this technology, the BRAM occupies 49% of the board-level resources after synthesis. It can be seen that this technology occupies fewer hardware resources while meeting the requirements of system design, providing an important foundation for further expansion of the system's functions. After the HDL design of the camera and HDMI display modules are completed, the IP packaging tool is exploited to uniformly package the design of these modules to facilitate subsequent IP reuse and transplantation. Key technology: video downsampling technology is adopted to reduce the size of the video frame buffer (BRAM) because the board-level storage capacity is only 13 Mb. The principle of this technology is that the adjacent pixels have similar values and the value can be restored through prediction even though some pixel values are discarded. The technology samples an effective pixel for every n pixels and writes it into BRAM ($n = 1, 3, \dots, 2n + 1$) during continuously generating pixel data by the camera. Among them, n is an odd number that facilitates subsequent data processing and prevents image display problems. The BRAM capacity processed by this method becomes the original $1/(n + 1)$ of the original, thus effectively reducing the consumption of storage resources. Though the BRAM capacity is reduced, the system still needs to output a whole frame with a resolution of 1024×768 , and it needs to exploit the average prediction method to restore the standard HDMI display output. Specifically, the HDMI display device reads the originally stored pixels from the BRAM. Taking the data of every $n + 1$ pixel as a group, the first datum of each group is the valid datum sampled and stored in the BRAM. The remaining n pixels need to be averaged to predict the output. This paper adopts the video downsampling method with $n = 3$. In this case, one valid datum is sampled from every three pixel data. Additionally, every four pixels of data are taken as a group and two groups of data are used for specific descriptions. The processing process is shown as follows. The output pixel data is

$$P_{00}, P_{01}, P_{02}, P_{03}, P_{10}, P_{11}, P_{12}, P_{13} \cdots \quad (1)$$

For pixel data P_{xy} , x is the group number and y is the number in the group. The first average processing is

$$P_{02} = P_{00}/2 + P_{10}/2. \quad (2)$$

The second average processing is

$$P_{01} = P_{00}/2 + P_{02}/2, P_{03} = P_{02}/2 + P_{10}/2. \quad (3)$$

The pixel data is in the format of RGB565 with a total of 16 bits. $R_{xy} = P_{xy}$ [15: 11], $G_{xy} = P_{xy}$ [10: 5], and $B_{xy} = P_{xy}$ [4: 0].

It should be noted that the video downsampling technology deals with binary data, and it is implemented with a hardware description language. As can be seen from the above description, the intermediate processing first averages the pixel data and then sums it up. The average operation is performed before the summation because the summation may cause overflow for binary data with a specified bit width, which loses the high-bit weight and distorts the pixel data after the average value. Therefore, to avoid the overflow problem, the pixel values of the R, G, and B channels are processed in order [23]. By adopting the above method, the capacity of a single BRAM becomes 1/4 of the original capacity (only 1/4 frame of a pixel datum is stored). In this case, the two BRAMs required by the system occupy a total of 6 Mb of the storage resources, and the video images can be collected and output. Therefore, under the condition of limited hardware resources, the use of the video downsampling technology can effectively reduce the storage resource needed by high-resolution devices, thus improving the utilization of board-level resources and providing a broad space for the optimization and expansion of development board functions. Additionally, the video sampling technology helps optimize application performance, greatly reduces the amount of data transmitted, and improves the neural network's efficiency for processing pixel data.

4. System Software Design and Implementation

The implementation of the system software mainly includes two aspects: the operating system and application program. In our hardware implementation, the SD card acts as external storage that is equivalent to a hard disk. The operating system needs to be transplanted to the partitioned SD card, and the transplantation [24] is composed of three parts: the transplantation of U-Boot, Linux kernel, and roofs.

4.1. Linux Operating System Transplantation

U-Boot: The U-Boot [25] is first configured and then compiled through the Makefile. Next, the bootloader [26] is started in the SD card to initialize DDR and other peripherals. Finally, the Linux kernel is copied from the SD card to the DDR to start the Linux kernel.

Linux-kernel: The Debian Linux 5.5.0 is selected as the Linux operating system kernel for transplant to the development board [27]. After the source code is downloaded, a configuration is required. Then, the operating system image file and the DTB file are generated and transplanted to the development board after cross-compilation. The DTB file is also called the device tree file, and it is a binary file obtained by compiling the DTS file, which is a file that describes device information on the development board.

Roofs: Roofs refers to the root file system of Debian, and it is used after the Linux kernel is created.

4.2. Linux Device Driver Programming

Linux device driver [28] refers to the device driver of the Linux system, and it is a hardware interface that allows computer software and hardware to interact. The operating system can use this interface to control the hardware devices. According to the system hardware implementation in this paper, the driver framework needs to be designed and the drivers for critical peripherals need to be written. As mentioned above, the AXI bus is responsible for the communication between various modules of the system. Considering the driver's reusability, separation, and layering, the platform driver framework is adopted and the traditional character device driver is exploited to facilitate the driver's programming. The platform driver framework consists of the bus model, device model, and driver model. The driver framework mainly includes driver entry and exit functions, module loading and unloading functions, device registration and deregistration functions, operation set, platform driver structure, and matching table. Among them, the set of operation functions for implementing basic operations on device files include `open()`, `release()`, `read()`, and `write()`. Since the Linux kernel of version 5.0 or higher is used in this paper, the Linux driver already supports the device tree, and the driver can control specific peripherals based on the device node information. In the device tree, each device node represents a specific

device and different devices have different properties. In addition to standard device attributes, users can customize attributes. These device attributes can then be used as items in the matching table to match the drivers to the devices. In this paper, two device nodes are designed under the SoC node for the key peripherals, and the attributes of “compatible” and “reg” are set. The former is a compatibility attribute for matching between drivers and peripherals, and the latter describes the device address space information, which is allocated to the device after the module is connected. The key peripherals in the system implementation include servo, camera, and HDMI display. The design of the servo’s driver uses the copy from the user() function and the memcpy() function to write the registers that control left and right movements of the dual-axis servo pan/tilt, and subsequent control of the servo is realized by the application program. The design of the HDMI’s driver reads the frame data from the BRAM through the copy provided by the user() function. Based on this, the data frames can be processed by OpenCV program for face detection and tracking [29].

4.3. Models and Application

4.3.1. Introduction of Models

The three object detection models that are provided by OpenCV [30–32] are used in this paper, namely the YOLO model, the Haar model, and the LBP model. YOLO: YOLO [33] is an object recognition and localization model based on deep neural networks. It uses a separate convolutional neural network model, and the input image is divided into $S \times S$ grids. To achieve end-to-end object detection, each cell is responsible for detecting the object for which the center point falls into the grid and gives the probability that the object belongs to different categories. YOLO has the characteristics of simplicity, high speed, and strong generalization ability. Haar: Haar is a cascaded classifier based on the Haar feature that reflects the changes in image grayscale [34]. Haar cascading is a method based on machine learning, and the cascading function can be trained through positive and negative images. OpenCV provides a cascaded object detector for Haar models [35]. This detector is featured with high computational efficiency and fast detection speed. LBP: LBP model is a cascaded classifier that can detect objects as well as the Haar model. As an effective texture operator, the LBP feature [36] can be combined with the histogram of LBP feature spectrum statistics to achieve real-time image classification and recognition [37]. OpenCV provides an LBP detector that extracts the LBP feature in the picture and uses the statistical histogram of the LBP feature spectrum as a feature vector for classification and recognition. LBP model has a fast training speed and good object detection capability.

4.3.2. Design of Application

Applications should be highly valued because they determine the function of the system. The design of applications mainly includes two parts. The first part is information interaction between the user space and the kernel space, and the second part is data processing of the transmitted information through different models. For the first part, the application is in the user space and the driver is in the kernel space. Since everything in the Linux system can be regarded as a file, the corresponding file is generated after the driver is successfully loaded. To control the hardware, the application program performs related operations on the driver file. These operations include API functions such as open(), close(), write(), and read(). They all correspond to related library functions. After a system call, the related drivers in the kernel are executed, and the corresponding hardware devices are controlled finally. In addition to opening and closing the driver’s basic operations, the registers of the servo module need to be written to control the servo’s movement in the horizontal and vertical directions. In contrast, the application of the HDMI module needs to read the BRAM IP to obtain the video data. The second part of the information processing is further divided into the following points. By calling the library file of OpenCV, the data frame collected from the camera to the array of the application program through the driver is stored and the three-channel image variable with a size equal to the pixel size of the data frame is created. By traversing the rows and columns of the

array stored in the data frame, the data of the three RGB channels are extracted. Then, the `CV::CascadeClassifier` class is exploited to detect the images collected from the video stream. Meanwhile, `CV::CascadeClassifier::load` is exploited to load a .XML classifier that can either be a Haar or LBP classifier. Moreover, `CV::CascadeClassifier::detectMultiScale()` is exploited to test the images of human faces obtained and the `getCenterPoint()` function is exploited to obtain the position of a face's geometric center. Different from the cascading classifier, the YOLO model for object detection requires invoking the DNN class provided by OpenCV and loading the model configuration file, weight file, and model category label file. Subsequently, the moving position of the object is determined by reading the continuous data frames and by processing the data of the object detection model. Based on the object position, the linear feedback PID algorithm [38] is used to calculate the moving distance of the servo in the horizontal and vertical directions. Next, the moving direction and distance are provided to the driver. In this way, real-time tracking and monitoring of moving objects can be achieved at the hardware level of the system. The execution time of the object detection model is the main part of the program running time. In this case, the choice of object detection model directly affects the system's performance. In the subsequent experiments, the above three models are implemented, and the optimal model is selected by comparing the results of these models to achieve better object detection and tracking.

5. Experiment

5.1. Process

The system integration is completed according to the previous description. Specifically, the Vivado design kit is exploited to implement the logic design, module connection, address allocation, and bitstream generation. Then, the data lines of the peripherals including the UART, USB, and ethernet are connected to the host's development board. Meanwhile, the display screen is connected to the HDMI interface, and the servo is connected to Pmod A; the OV5640 camera is connected to Pmod B and C. After that, the bitstream is programmed to the FPGA of the Nexys Video development board to implement the RISC-V processor core. After the device tree file is modified and compiled, the generated U-Boot, operating system image file, and root file system are transplanted to the SD card, and the operating system runs successfully. Subsequently, the compiled driver and program are transferred to the file system via ethernet. After the driver is loaded and the application is run, the results of object detection and tracking are observed. Finally, the image of the tracked and monitored objects is uploaded to the cloud server, and it is compared with the suspect in the database through face recognition. Additionally, the relevant situation is reported to the security management department in time. The final experimental operating environment is shown in Figure 4.

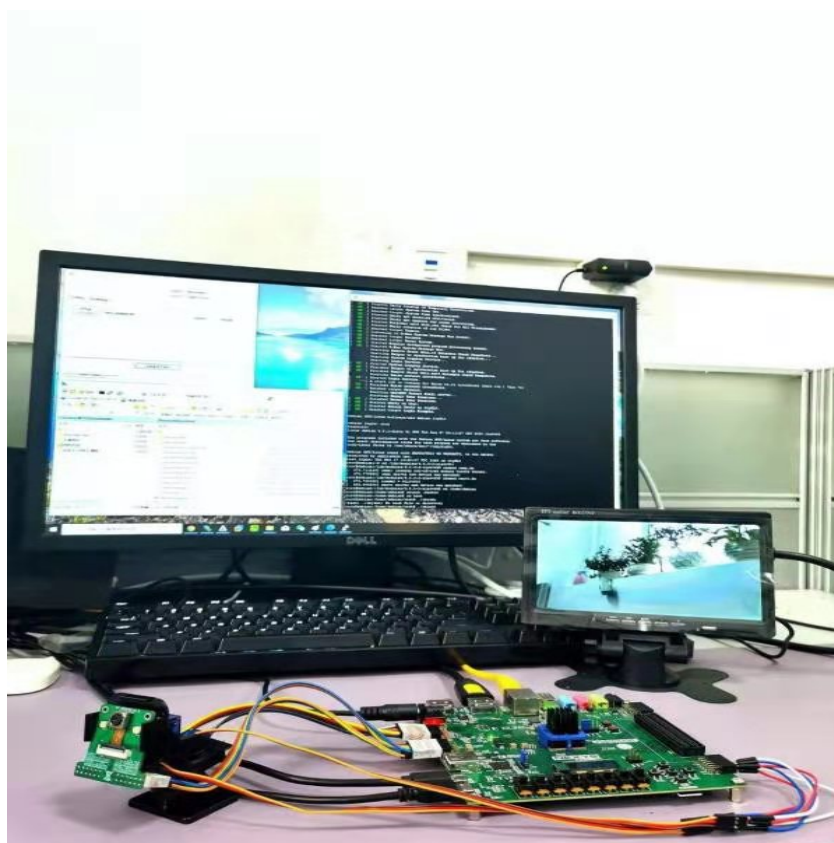


Figure 4. Experimental operating environment.

5.2. Results and Discussion

An intelligent security monitoring system aims to detect and track moving objects within a specific area. The functions of the system designed and implemented in this paper are inevitably restricted by the hardware resources and the selected model. To obtain better functions of the system, the experiment presents the hardware resource utilization and the performance comparison results of different models. The hardware resource utilization is listed in Table 1. It can be seen that, after the use of video sampling technology, the storage resource utilization of BRAM is 56.71%.

Table 1. Resource costs and utilization.

Resource	Utilization	Available	Utilization %
LUT	85,371	134,600	63.43
LUTRAM	5847	46,200	12.66
FF	50732	269,200	18.85
BRAM	207	365	56.71
DSP	30	740	4.05
IO	102	285	35.79
MMCM	2	10	20.00
PLL	2	10	20.00

Since object detection is based on the neural network model and is affected by the characteristics of the model, the object detection time accounts for the main part of the application running time. Experiments are conducted to measure the detection time of the three models for processing a single image. The experimental results are listed in Table 2.

Table 2. The detection time of different models.

Model	Detection Time/Frame (s)
YOLO v4 tiny	403.14
Haar	5.37
LBP	1.25

The YOLO model is a well-known object detection model and is widely used in computer vision. Thus, the latest and lightest YOLO v4 tiny model was used in the experiment. However, the single-frame detection time of this model is 403.14 s, which is worse than other models and fails to achieve the real-time performance required by the system design. The analysis of the result indicates that the hardware resources make the system performance limited by the processor's dominant frequency. Therefore, the Haar and LBP face detection models were used in the subsequent experiments. By converting the detection of multiple objects to the detection of faces, the neural network is simplified, and the learning and training times are also significantly reduced. Meanwhile, a model for face detection without sacrificing the effective range of object detection is desired. Especially, the smallest and largest rectangular windows suitable for object detection are determined through experiments, further reducing the object detection time. For the face detection model, the classic Haar model and the LBP model were compared. According to the experimental results, the detection time of the LBP model for a single frame is 1.25 s, while the Haar model requires 5.37 s to detect a single frame. The LBP model performed better than the Haar model, so the LBP model was chosen to realize real-time tracking and monitoring of moving objects. Overall, the intelligent security monitoring system studied in this paper can meet the requirements of detection and tracking of moving objects. However, some shortcomings need to be overcome by future work. In terms of the hardware, the hardware design based on the embedded platform is limited in resources, and the improvement and expansion of the functions require an improved utilization of the hardware resources. Additionally, the hardware accelerators can be designed specifically for the detection and tracking algorithm to further improve the performance of the system. In terms of the software, the design of the driver and application to realize object detection by calling the models of Haar, LBP, and Yolo were finished independently. In the future, independent training of the models considering the characteristics of the embedded platform can be conducted to meet the performance requirements of deploying the algorithm on terminals and to further improve the detection accuracy and speed.

6. Conclusions

To construct an intelligent security monitoring system, the RISC-V SoC hardware platform was designed and built in this paper and the Linux operating system was transplanted. Moreover, the drivers and application programs are designed so that the processor can control the peripheral devices. According to the comparison between the three models, the application program based on the LBP model was chosen to realize real-time object detection and tracking. In addition, a video downsampling technology was adopted by this paper. In the case of limited hardware resources, this technology ensures that the video data are output successfully and that the system performance was improved effectively by reducing data processing for one frame. In addition, the research of this paper still has some limitations. The design is implemented on an FPGA chip because it can quickly adapt to the changing application requirements owing to the flexibility and programmability of FPGA. However, for practical applications of intelligent security monitoring, design and implementation on an FPGA requires a larger area, higher power consumption, and lower operating frequency than that on the ASIC. Meanwhile, since RISC-V is an emerging open-source instruction set, its ecology system is not currently perfect. The documentation for developers is scarce, the development environment is at a low level of maturity, and debugging is difficult. However, with increasing development of the RISC-V ecol-

ogy system, the research in this paper can also be further improved and promoted. The system performance can be improved from two aspects. Considering the hardware, the development board with more hardware resources can be used. Additionally, due to the limitation of the processor's dominant frequency, processor cores with higher frequency can be used. Regarding the software, the program can be further improved by optimizing models and the specific neural network can be designed independently for learning and training. Supporting by robust cloud storage, cloud computing, and other technologies, the application scenarios of the intelligent security monitoring system can be extended to social security. Based on this, the face of a criminal can be accurately detected and identified, and the driving trajectory of the person can be tracked and recorded, thus ensuring the security and stability of the whole society.

Author Contributions: Methodology, W.W.; formal analysis, D.S.; resources, B.Y.; writing—review and editing, Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key Research and Development Program of China (No. 2018YFB0204301). It was also funded by the Science and Technology Program of Hunan Province (No. 2018XK2102).

Data Availability Statement: The raw/processed data required to reproduce these findings cannot be shared at this time as the data also forms part of an ongoing study.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Verma, R.K.; Singh, P.; Panigrahi, C.R.; Pati, B. ISS: Intelligent Security System Using Facial Recognition. In *Progress in Advanced Computing and Intelligent Engineering*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 96–101.
2. Alheeti, K.M.A.; McDonald-Maier, K. An Intelligent Security System for Autonomous Cars Based On Infrared sensors. In Proceedings of the 2017 23rd International Conference on Automation and Computing (ICAC), Huddersfield, UK, 7–8 September 2017; pp. 1–5.
3. Yerragolla, M.; Pallela, K.; Gera, I.P. Intelligent Security System for Residential and Industrial Automation. In Proceedings of the 2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON), Varanasi, India, 9–11 December 2016; pp. 229–234.
4. RenukaChumurkar, M.; Bagdi, V. Smart Surveillance Security & Monitoring System Using Raspberry PI and PIR Sensor. *Int. J. Sci. Eng. Appl. Sci. IJSEAS* **2016**, *2*, 1–4.
5. Menezes, V.; Patchava, V.; Gupta, M.S.D. Surveillance and Monitoring System Using Raspberry Pi and SimpleCV. In Proceedings of the 2015 international conference on green computing and internet of things (ICGCIoT), Greater Noida, India, 8–10 October 2015; pp. 1276–1278.
6. Othman, N.A.; Aydin, I. A New IoT Combined Body Detection of People by Using Computer Vision for Security Application. In Proceedings of the 2017 9th International Conference on Computational Intelligence and Communication Networks (CICN), Girne, Northern Cyprus, Turkey, 16–17 September 2017; pp. 108–112.
7. Aydin, I.; Othman, N.A. A New IoT Combined Face Detection of People by Using Computer Vision for Security Application. In Proceedings of the 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, Turkey, 16–17 September 2017; pp. 1–6.
8. Wu, L.; Liu, S. Comparative Analysis and Application of LBP Face Image Recognition Algorithms. *Int. J. Commun. Syst.* **2021**, *34*, e3977. [[CrossRef](#)]
9. Othman, N.A.; Salur, M.U.; Karakose, M.; Aydin, I. An Embedded Real-Time Object Detection and Measurement of Its Size. In Proceedings of the 2018 International Conference on Artificial Intelligence and Data Processing (IDAP), Malatya, Turkey, 28–30 September 2018; pp. 1–4.
10. Kanter, D. RISC-V Offers Simple, Modular ISA. In Microprocessor Report, 2016. Available online: <https://riscv.org/wp-content/uploads/2016/04/RISC-V-Offers-Simple-Modular-ISA.pdf> (accessed on 20 August 2020).
11. Waterman, A.S. Design of the RISC-V Instruction Set Architecture. Ph.D. Thesis, UC Berkeley, Berkeley, CA, USA, 2016.
12. Patterson, D.; Waterman, A. *The RISC-V Reader: An Open Architecture Atlas*; Strawberry Canyon LLC, San Francisco, CA, USA, 2017.
13. Farshchi, F.; Huang, Q.; Yun, H. Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim. In Proceedings of the 2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), Washington, DC, USA, 17 February 2019; pp. 21–25.

14. Flamand, E.; Rossi, D.; Conti, F.; Loi, I.; Pullini, A.; Rotenberg, F.; Benini, L. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In Proceedings of the 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP), Milan, Italy, 10–12 July 2018; pp. 1–4.
15. Zhong, X.; Sham, C.W.; Ma, L. A RISC-V SoC for Mobile Payment Based on Visible Light Communication. In Proceedings of the 2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), Ha Long, Vietnam, 8–10 December 2020; pp. 102–105.
16. Bailey, S.; Rigge, P.; Han, J.; Lin, R.; Chang, E.Y.; Mao, H.; Wang, Z.; Markley, C.; Izraelevitz, A.M.; Wang, A.; et al. A Mixed-Signal RISC-V Signal Analysis SoC Generator with a 16-nm Finfet Instance. *IEEE J. Solid-State Circuits* **2019**, *54*, 2786–2801. [\[CrossRef\]](#)
17. Torres-Sánchez, E.; Alastruey-Benedé, J.; Torres-Moreno, E. Developing an AI IoT Application with Open Software on a RISC-V SoC. In Proceedings of the 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS), Segovia, Spain, 18–20 November 2020; pp. 1–6.
18. Höller, R.; Haselberger, D.; Ballek, D.; Rössler, P.; Krapfenbauer, M.; Linauer, M. Open-source RISC-V Processor IP Cores for FPGAs—Overview and Evaluation. In Proceedings of the 2019 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 10–14 June 2019; pp. 1–6.
19. Menon, A.; Murugan, S.; Rebeiro, C.; Gala, N.; Veezhinathan, K. Shakti-T: A RISC-V Processor with Light Weight Security Extensions. In Proceedings of the Hardware and Architectural Support for Security and Privacy, Toronto, ON, Canada, 18 June 2017; pp. 1–8.
20. Matthews, E.; Shannon, L. TAIGA: A New RISC-V Soft-Processor Framework Enabling High Performance CPU Architectural Features. In Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 4–8 September 2017; pp. 1–4.
21. Balkind, J.; Lim, K.; Gao, F.; Tu, J.; Wentzlaff, D.; Schaffner, M.; Zaruba, F.; Benini, L. OpenPiton+ Ariane: The First Open-Source, SMP Linux-booting RISC-V System Scaling From One to Many Cores. In Proceedings of the Workshop on Computer Architecture Research with RISC-V (CARRV), Phoenix, AZ, USA, 22 June 2019; pp. 1–6.
22. Asanovic, K.; Avizienis, R.; Bachrach, J.; Beamer, S.; Biancolin, D.; Celio, C.; Cook, H.; Dabbelt, D.; Hauser, J.; Izraelevitz, A.; et al. The Rocket Chip Generator. Tech. Rep. UCB/EECS-2016-17. 2016. Available online: <https://www2.eecs.berkeley.edu/Pubs/Tech\Rppts/2016/EECS-2016-17.html> (accessed on 16 August 2020).
23. Chen, L.; Hu, M.; Liu, N.; Zhai, G.; Yang, S.X. Collaborative Use of RGB and Thermal Imaging for Remote Breathing Rate Measurement under Realistic Conditions. *Infrared Phys. Technol.* **2020**, *111*, 103504. [\[CrossRef\]](#)
24. Yanzhong, Z.; Yanyan, C. Study of Some Common Technology on the Design and Transplantation of Embedded Linux Operating System. *Comput. Autom. Meas. Control* **2005**, *2*, 162–164.
25. C15f44a. u-Boot. <https://github.com/u-boot/u-boot> (accessed on 23 December 2020).
26. Mao, H. RISC-V Proxy Kernel and Boot Loader. <https://github.com/riscv/riscv-pk> (accessed on 27 August 2020).
27. Debian Linux Package Repositories. <https://wiki.debian.org/RISC-V> (accessed on 10 October 2020).
28. Kadav, A.; Swift, M. Understanding modern device drivers. *ACM SIGPLAN Notices* **2012**, *47*, 87–98. [\[CrossRef\]](#)
29. Zhou, J.; Pun, C.M. Personal Privacy Protection via Irrelevant Faces Tracking and Pixelation in Video Live Streaming. *IEEE Trans. Inf. Forensics Secur.* **2020**, *16*, 1088–1103. [\[CrossRef\]](#)
30. Xu, Z.; Baojie, X.; Guoxin, W. Canny Edge Detection based on OpenCV. In Proceedings of the 2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), Yangzhou, China, 20–22 October 2017; pp. 53–56.
31. Pulli, K.; Baksheev, A.; Korniyakov, K.; Eruhimov, V. Real-time Computer Vision with OpenCV. *Commun. ACM* **2012**, *55*, 61–69. [\[CrossRef\]](#)
32. Bradski, G.; Kaehler, A. *Learning OpenCV: Computer Vision with the OpenCV Library*; O'Reilly Media, Inc.: Cambridge, MA, USA, 2008.
33. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
34. Lienhart, R.; Maydt, J. An Extended Set of Haar-like Features for Rapid Object Detection. In Proceedings of the International Conference on Image Processing, Rochester, NY, USA, 22–25 September 2002; Volume 1, p. I.
35. Team, O.D. Face Detection Using Haar Cascades. [Tersedia:http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html](http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html) (accessed on 26 May 2015).
36. Zhang, G.; Huang, X.; Li, S.Z.; Wang, Y.; Wu, X. Boosting Local Binary Pattern (LBP)-based Face Recognition. In *Chinese Conference on Biometric Recognition*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 179–186.
37. do Prado, K.S. Face Recognition: Understanding LBPH Algorithm. <https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b> (accessed on 24 September 2020).
38. Li, Y.; Ang, K.H.; Chong, G.C. PID control system analysis and design. *IEEE Control Syst. Mag.* **2006**, *26*, 32–41.