SMART DOOR LOCK SYSTEM BASED ON STM32 AND K210

Bowen Zhang

3rd Year Project Final Report

Department of Electronic &
Electrical Engineering

UCL

Supervisor: Prof. Paul Brennan

5 April 2025

I have read and understood UCL's and the Department's statements and guidelines concerning plagiarism.

I declare that all material described in this report is my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures and computer programs.

I acknowledge the use of Chat GPT-4o (OpenAI, https://chat.openai.com) to check grammar mistake and translation from Chinese to English.

This report contains 44 pages and 9511 words

Signed: _____Bowen ZHANG_____ Date: ___09.04.2025___

A Smart door lock system based on stm32 and k210

Table of Contents

1 Abstract 5
2 Introduction 6
3 Literature Review 7
4 Methods 8
4.1 Hardware Selection 8
4.2 Implementation of 2.42-Inch OLED Screen 11
4.3 Implementation of 4*4 Keypad Matrix 12
4.4 Menu Design and Programming 13
4.5 Implementation of Remote Unlocking Function 16
4.5.1 HC-05 Bluetooth Module Setup 16
4.5.2 STM32 Program 17
4.6 Implementation of Fingerprint Unlocking Function 18
4.6.1 AS608 Hardware Setup 18
4.6.2 AS608 Functions Introduction 18
4.6.3 Software Implementation of Fingerprint Authentication 19
4.7 Implementation of Facial Recognition Function 25
4.7.1 Maix Bit Hardware Setup 25
4.7.2 Maix Bit Software Implementation 26
4.7.3 STM32 Software Implementation 30
4.8 Implementation of the Power Supply Module 31
4.9 Implementation of the Custom PCB 32
4.10 Workflow Introduction 33
5 Results, analysis, and discussion 34
5.1 Overview of the Whole System 34
5.2 Unlocking through Bluetooth (Remote Unlocking) 35
5.3 Unlocking through PIN + Fingerprint Verification 35
5.3.1 PIN Verification 35
5.3.2 Fingerprint Verification 36
5.3.3 Modifying Fingerprint Data 36
5.4 Unlocking through PIN + Facial Recognition 38
5.4.1 Facial Verification 38
5.4.2 Modifying Facial Data 38

5.5 Limitations and Possible Solutions 39
5.5.1 Limitations in Keypad Matrix 39
5.5.2 Limitations in HC-05 Bluetooth Module 39
5.5.3 Limitations in Power Supply Module 40
6 Conclusions and future work 41
6.1 Conclusions 41
6.2 Future Work 41
7 References 43
8 Appendices 44

1 Abstract

This report presents the design and implementation of a smart door lock. It integrates multiple unlocking methods: fingerprint recognition, facial recognition, keypad input and remote unlocking, using an STM32 microcontroller and a K210 development board. A custom PCB was designed and fabricated to consolidate the system's core components, ensuring robust functionality, and support future streamlined manufacturing. The STM32 microcontroller is used in this project as a central controller, it interfaces with fingerprint sensor, Bluetooth module, keypad, OLED screen, and also communicate with K210. The K210 board is used for executing facial recognition tasks, it will communicate with STM32 microcontroller and send results of the facial data matching to STM32. This work not only demonstrates an affordable solution for enhanced residential security but also provides a scalable hardware and software framework for future expansion.

2 Introduction

Traditional mechanical door locks can no longer meet the security needs of modern life. With advances in technology, physical keys are easy to duplicate, and mechanical locks are easy to break into[2]. A much safer and efficient way is to incorporate the door locks with the residents of the building biometric data which may include: finger prints, iris, or face among others[1]. These are known as smart locks. However, most of the smart door locks on sale today are based on detecting fingerprints or entering a code to unlock the door, which is not friendly to people who are unable to use their fingerprints (e.g., finger injuries). Furthermore, because smart locks are expensive, they are mainly used in places like banks and laboratories rather than residential buildings.

This project aims to create a low-cost smart door lock that is suitable for residential use. The smart lock will support multiple unlocking methods, including fingerprint recognition, facial recognition, keypad input, and remote unlocking. To unlock, use must first enter correct password then verify their biometric data.

The STM32 microcontroller will act as the main processor, managing the keypad, fingerprint recognition, and remote unlocking. The K210 development board will handle facial recognition, as it is designed specifically for this purpose. This project also involves assembling and making a PCB of our own to integrate the components such as the STM32, fingerprint sensor, keypad, Wi-Fi module and K210 board. The PCB design is aimed to achieve reliability and efficiency of the system.

3 Literature Review

In modern life, the usage of smart door lock systems has become even more crucial for managing the current issues. Previous approaches including PCA have been common because of their efficient computation and ease of implementation. However, their robustness is constrained and performs poorly in environments with different illumination and complicated backgrounds. Hierarchical networks which include deep learning architectures such as ResNet101 and FaceNet have been found to be more accurate and less susceptible to spoofing attacks thus making them ideal for facial recognition[1][5].

Combining multiple biometric systems like fingerprints and face identification increases the security and minimises the risk of using a single system. Integrated boards such as Raspberry have shown to be useful in coordinating image acquisition and identification processes, and are portable and consume less power. However, the ability to adapt to the environment to adapt for instance, to light conditions is still a problem[2][5].

Smart locks incorporate new features through connectivity with IoT such as real time monitoring, remote control and intruder alarms. The simplicity associated with Bluetooth and cloud data bases for access control has its limitations in range and scalability. It has been proposed that the use of Wi-Fi for remote monitoring is more effective than the current method[3].

To strengthen security, the technology of the blockchain has been used to avoid any unauthorized access and data manipulation. Smart door lock systems have been made more reliable by the use of blockchain systems for storage and authentication[4].

This project extends these developments by combining STM32 and K210 to implement an affordable smart door lock system. Multi-modal biometric authentication, IoT for monitoring, and the design of the hardware are the solutions that meet the current challenges and improve the system's accessibility and stability.

4 Methods

4.1 Hardware Selection

The objective of this project was to develop a low-cost yet robust smart door lock system that integrates multiple authentication methods. To meet this goal, hardware components were carefully selected and, in some instances, replaced based on availability, performance requirements, and cost considerations:

- **Main Microcontroller: STM32F103C8T6 (“Blue Pill”)**

The STM32F103C8T6, which is also known as the “Blue Pill,” is selected for its combination of affordability, solid performance, and large community support base. Priced at roughly £1 per unit, it features a 32-bit ARM Cortex-M3 processor that delivers sufficient computational power for handling keypad inputs, fingerprint authentication, and Bluetooth communication. Additionally, it offers flexible General-Purpose Input/Output (GPIO) options and integrated hardware peripherals for smooth interfacing with various sensors and modules. This balance of low cost and capable performance makes the “Blue Pill” an ideal choice for the core controller of the smart door lock system.

Price: about £1 per unit.

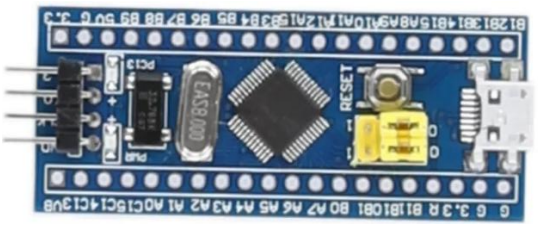


***Figure 4.1.1**: Real Time Image of STM32F103C8T6*

- **Keypad Matrix:**

The 4×4 membrane keypad matrix serves as the primary user input interface for the smart door lock system. It allows efficient PIN entry, menu navigation, and general control through a simple row-and-column scanning mechanism. Each key press is detected by driving one column at a time while reading the states of the rows, which significantly reduces the I/O overhead on the microcontroller. Additionally, the low-cost membrane design (around £0.5 per unit) offers durability and straightforward integration, making it suitable for embedded applications that require reliable keypad inputs.

Price: about £0.5 per unit.

***Figure 4.1.2**: Real Time Image of the 4*4 Keypad Matrix*

- **2.42-Inch OLED Display:**

The SSD1309-driven black-and-white OLED screen was chosen for its clarity and compatibility with the I2C communication protocol. It is used to display system menus and real-time status. In the preliminary assessment of the project, there was also the option of using a 0.96-inch OLED display based on the SSD1306 chip. Nevertheless, the screen may turn out to be small, which will lead to poor or limited display of information, not very suitable for real life use. The clarity and usability of real-time data presentation are critical in ensuring the effectiveness of user interfaces in embedded systems[2]. For this reason, the project went for the larger but costly 2.42-inch OLED screen since it provides better readouts and usability.

Price: about £6.5 per unit.

Figure 4.1.3*: Real Time Image of the 2.42-inch OLED Screen*

- **Fingerprint Unlocking Module**

The AS608 fingerprint sensor has been selected for this project due to its reliable performance, compact form factor, and integrated image-processing capabilities. It communicates via a simple UART interface, making it straightforward to connect with a microcontroller like the STM32F103C8T6. In addition, the AS608 offers robust matching algorithms and rapid enrolment times, contributing to both accuracy and convenience in user authentication workflows. This balance of stable performance, ease of integration, and moderate cost makes the AS608 a suitable choice for implementing fingerprint-based access control in a smart door lock system.

Price: about £5 per unit

Figure 4.1.4*: Real Time Image of the 2.42-inch OLED Screen*

- **Remote Unlocking Module**

The HC-05 Bluetooth module provides a simple and effective way to enable remote unlocking of the smart door lock system. By communicating with the STM32F103C8T6 through a UART interface, it allows users to control or monitor the lock from a smartphone or other Bluetooth-enabled device within a typical indoor range. Its straightforward configuration process, combined with broad availability and cost-effectiveness, makes the HC-05 a practical choice for adding remote access functionality.

Price: about £6.5 per unit.

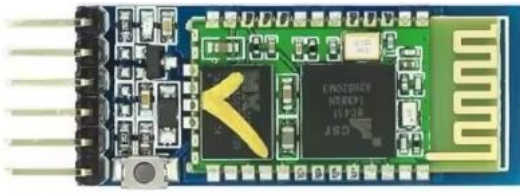


Figure 4.1.5*: Real Time Image of the HC-05 Bluetooth Module*

- **Facial Recognition Module**

The Maix Bit K210 development board was used to carry on the facial recognition function in this project. Maix Bit was powered by a dual-core 64-bit RISC-V processor with integrated hardware accelerators optimized for AI tasks. Specifically designed for real-time image processing, the Maix Bit efficiently handles face detection and recognition tasks locally without burdening the main microcontroller (STM32F103C8T6). Additionally, it connects seamlessly to a compact camera module for image acquisition and a 2.42-inch full-colour OLED screen for instant visual feedback, allowing users to clearly view recognition results. The Maix Bit's small form factor, low cost, real-time processing capability, and easy integration make it ideal for implementing accurate and responsive facial authentication in smart door lock systems.

Price: about £20 per unit.

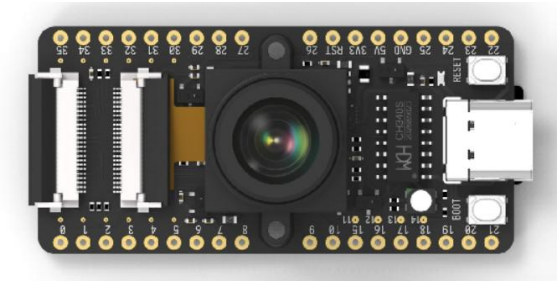


Figure 4.1.6*: Real Time Image of the Maix Bit K210 Development Board with Camera*

- **Power Supply Module**

The power supply for the system uses two MB102 power supply modules, each connected to a PP3 battery, providing both 3.3V and 5V outputs. This dual voltage setup ensures that components such as the STM32 microcontroller, fingerprint sensor, and OLED display receive the correct power. The MB102 modules are compact, cost-effective (around £1.5 per unit), and offer reliable performance for this project, making them an ideal choice for the smart door lock system.

Price: about £0.6 per unit.

***Figure 4.1.6**: Real Time Image of the MB102 DC-DC module and PP3 battery*

4.2 Implementation of 2.42-Inch OLED Screen

The OLED screen used in this project is a 2.42-inch black-and-white display driven by the SSD1309 controller, chosen for its clarity and compatibility with the I2C communication protocol. This screen serves as a critical component of the system, providing real-time feedback, displaying menus, and showing status updates. The screen is connected to the STM32 microcontroller via four pins: VCC, GND, SCL, and SDA. The specific pin connections between the 2.42-inch OLED screen and the STM32 microcontroller are detailed in Table 5.2:

***Table 4.2**: Connections Between OLED and STM32*

OLED Pins	STM32 Pins
GND	GND
VDD	3.3V
SCL	PB6
SDA	PB7

The SSD1309 controller was initialized using a sequence of commands sent via the I2C protocol. The initialization process involves configuring memory addressing modes, setting contrast levels, and enabling the display. The ‘ssd1306_Init()’ function (compatible with SSD1309), adapted from an open-source library, played a key role in ensuring the OLED screen's operation. Commands were sent to define settings such as horizontal addressing mode, multiplex ratio, and segment remapping, ensuring optimal display functionality. Additionally, the system initializes with a screen-clearing command, ensuring a clean slate for rendering content[6].

The software development for the OLED integration also involved managing the display buffer, which serves as a temporary memory space to hold screen content before it is rendered. Functions such as ‘ssd1306_WriteString()’ and ‘ssd1306_UpdateScreen()’ were employed to render text and update the display in real time[6]. These features are particularly important for creating an interactive and responsive user interface. The OLED screen provides real-time system feedback, clearly displaying user menus, system status, and authentication results

4.3 Implementation of 4*4 Keypad Matrix

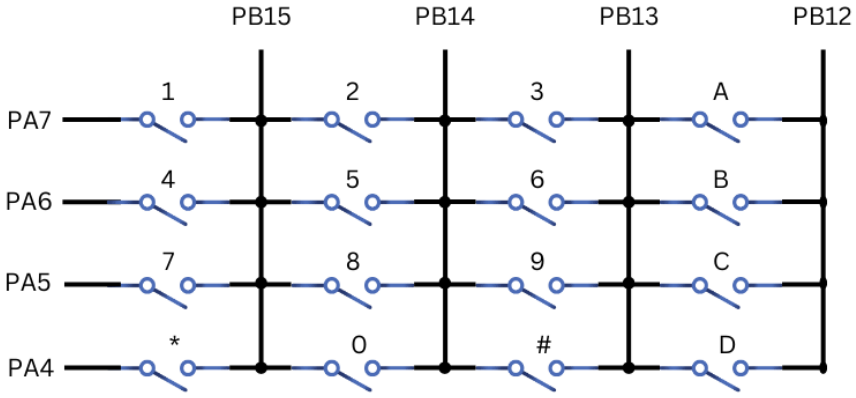


***Figure 4.3**: Connection Schematic of the 4*4 Keypad.*

The 4×4 keypad matrix is used as the main input device for the smart lock system, interfaced directly with the STM32 microcontroller's General-Purpose Input/Output (GPIO) ports. The keypad consists of 16 buttons arranged in a 4×4 grid, with rows connected to pins PA4, PA5, PA6, and PA7, and columns connected to pins PB12, PB13, PB14, and PB15. The row pins are configured as inputs, while the column pins serve as outputs, enabling the detection of key presses using a simple row-column scanning method.

The scanning mechanism operates by sequentially driving each column high while keeping others low and reading the states of all row inputs. For example, to detect keys in the first column, PB12 is set high, while PB13, PB14, and PB15 are set low. The system then scans the states of row pins (PA4, PA5, PA6, and PA7) to determine which specific key has been pressed. This process repeats for each column pin (PB13, PB14, PB15).

During scanning, each detected key press is assigned a specific ASCII code (e.g., the button labeled '1' corresponds to ASCII 49, and the button 'A' corresponds to ASCII 65), and the detected value is stored in a global variable named keyPressed.

Listing 4.3.1: An Example of Keypad Scanning Logic in ISR

```
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, 0);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, 0);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, 0);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, 1);
if(GPIO_Pin == GPIO_PIN_4 && HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_4))
{
  keyPressed = 68; // ASCII value of D
}
else if(GPIO_Pin == GPIO_PIN_5 && HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5))
{
  keyPressed = 67; // ASCII value of C
}
else if(GPIO_Pin == GPIO_PIN_6 && HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_6))
{
  keyPressed = 66; // ASCII value of B
}
```

```
else if(GPIO_Pin == GPIO_PIN_7 && HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_7))
{
  keyPressed = 65; // ASCII value of A
}
```

An interrupt-driven approach is employed to enhance the responsiveness of keypad detection. When a key is pressed, an interrupt triggered by a rising-edge signal from the row GPIO pins activates the interrupt service routine (ISR). To ensure accurate detection and prevent multiple triggers from a single press, software debouncing is implemented by comparing timestamps (HAL_GetTick()) and accepting a key press only if 200 ms have elapsed since the previous detection. The ISR then sequentially drives each column pin high and reads the states of corresponding row pins to identify the pressed key. After the pressed key is successfully identified, GPIO configurations are restored, interrupt flags are cleared, and the system resets to handle subsequent key presses.

Listing 4.3.2: ISR Settings with Software Debouncing

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  currentMillis = HAL_GetTick();
  if (currentMillis - previousMillis > 200) {
    // Configure GPIO pins : PA4 PA5 PA6 PA7 to GPIO_INPUT
    GPIO_InitStructPrivate.Pin = GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
    GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
    GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
    GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStructPrivate);

    // Scanning Logic Mentioned Before

    // Configure GPIO pins : PA4 PA5 PA6 PA7 back to EXTI
    GPIO_InitStructPrivate.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStructPrivate.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStructPrivate);
    previousMillis = currentMillis;
  }
}
```

4.4 Menu Design and Programming

The goal for this project's menu system was to set up a user-friendly interface that can enable effective operations of the smart lock system. The menu is shown on the OLED screen and controlled by a 4*4 keypad matrix. This interface allows the user to move between various functions, in this stage, locks and unlocks the door, access system settings and modify the PIN to access the door.

The menu system operates as a state machine, where each menu screen represents a distinct state. This is managed in the 'main.c' file where the control flow is held with a 'flag' variable to show the current state of menus. These keys are on the first menu used to either unlock the door or to go to the setting menu by pressing certain keys. Moving between the states is done

via various functions like ‘initial_menu()’, ‘menu_pin()’, ‘menu_settings()’ as to maintain the correct flow these functions are defined in order.

Listing 4.4.1: Main Menu State Machine Implementation

```
switch (flag) {
    case 0: // Initial menu
        flag = initial_menu();
        break;

    case 1: // Menu to check Door PIN
        flag = menu_pin();
        break;

    case 11: // Menu to check Door PIN
        flag = unlock_method();
        break;

    case 12: // Menu to check Door PIN
        flag = menu_fingerprint_check();
        break;

    case 13:
        HAL_UART_Receive_IT(&huart1,&RX_dat,1);//turn on the interrupt of
uart2
        facial_flag = 0;
        flag = menu_facial_check();
        break;

    case 2: // Menu to check System PIN
        flag = menu_settings_check();
        break;

    case 3: // Menu to choose to edit system settings
        flag = menu_settings();
        break;

    case 31: // To change the Door PIN
        flag = set_doorPIN();
        break;

    case 32:
        flag = menu_modify_FR();
        break;

    case 33:
        flag = menu_record_FR();
        break;
```

```
    case 34:
        flag = menu_delete_FR();
        break;

    case 35:
        flag = menu_modify_Facial();
        break;

    default:
        flag = 0; // In case unexpected value occurs, go back to main menu
        break;
      }
```

The system initializes the OLED display and detects key inputs. In ‘menu.c’, functions like ‘ssd1306_Fill()’ and ‘ssd1306_WriteString()’ render text and instructions on the screen. For instance, ‘initial_menu()’ displays options to proceed or access settings. User input is detected via the keypad, with keys mapped to specific ASCII values. The program continuously monitors key presses and updates the `flag` variable accordingly to navigate between menu states.

Listing 4.4.1: An Example of Initial Menu Showing How the Functions Work

```
int initial_menu(void)
{
    static uint32_t delayStartTime = 0; // Stores the start time for non-
blocking delay
    static int delayInProgress = 0; // Indicates if a non-blocking delay is
active

    if (!delayInProgress) { // Only update screen when not in a delay period
        ssd1306_Fill(Black);
        ssd1306_SetCursor(11, 10);
        ssd1306_WriteString("Smart Door Lock", Font_7x10, White);
        ssd1306_SetCursor(0, 30);
        ssd1306_WriteString("Press C to Continue", Font_6x8, White);
        ssd1306_SetCursor(0, 40);
        ssd1306_WriteString("Press A to Settings", Font_6x8, White);
        ssd1306_UpdateScreen();
    }

    if (delayInProgress) {
        // **Non-blocking delay: Wait for 2 seconds**
        if (HAL_GetTick() - delayStartTime >= 2000) {
            delayInProgress = 0; // End delay
            BT_Flag = 0; // Reset Bluetooth unlock flag
            return 0; // Return to the main menu
        }
        return 0; // Stay in the current menu during the delay period
    }
```

```
    if (keyPressed == 'C') { // 'C' key pressed
        keyPressed = 0;
        return 1; // Switch to the next menu
    } else if (keyPressed == 'A') { // 'A' key pressed
        keyPressed = 0;
        return 2; // Switch to settings menu
    } else if (BT_Flag == 1) { // Bluetooth unlock triggered
        ssd1306_Fill(Black);
        ssd1306_SetCursor(0, 0);
        ssd1306_WriteString("Unlock!", Font_7x10, White);
        ssd1306_UpdateScreen();

        delayStartTime = HAL_GetTick(); // Record start time
        delayInProgress = 1; // Activate non-blocking delay
    }

    return 0; // Stay in the initial menu
}
```

4.5 Implementation of Remote Unlocking Function

This section introduces the remote unlocking feature through a Bluetooth connection. In this project, the remote unlocking function is realized through the **HC-05** Bluetooth module. When a valid unlock command is received via the UART interface (in this case, USART2), the system will set a flag to trigger unlocking. Users can send specific characters (for example, the letter **'o'**) from a connected Bluetooth device to prompt the door lock to open.

4.5.1 HC-05 Bluetooth Module Setup

The HC-05 Bluetooth module is a low-cost, widely used module for short-range Bluetooth communication. It operates via UART (Universal Asynchronous Receiver-Transmitter) and can easily be interfaced with microcontrollers such as the STM32. Before using it for remote unlocking, the module must be properly configured and connected.

Furthermore, to make the module good to use, some changes should be made to the HC-05 before connecting with STM32. Before making any changes, the module should be firstly set into AT mode. To access AT mode, simply connect KEY pin of the module to high before powering the module. After that, the module could be connected to a laptop through USB – TTL wire using serial communication software.

After successfully connecting the module with our laptop, these commands should be sent:

- 'AT': this command is used to test the communication
- 'AT+UART=9600,0,0': this command is used to set the baud rate to 9600
- 'AT+ROLE=0': this command is used to set the module as a slave
- 'AT+password=1234': this command is used to set the PIN to communicate with the module to 1234

To use the HC-05 Bluetooth module with the STM32 microcontroller, the connection between the module and STM32 microcontroller should be made as shown below:

Table 4.5: Connections Between HC-05 and STM32

HC-05 Pins	STM32 Pins
VCC	5V
GND	GND
TX	PA3 (UART2_RX)
RX	PA2 (UART_2 TX)

After configuration, the module could be connected to the can be paired with a smartphone or PC using the PIN set before.

4.5.2 STM32 Program

Once the hardware setup is complete, the next step is to integrate the HC-05 Bluetooth module into the STM32 firmware. The key part of the implementation is handling the incoming Bluetooth data and triggering the unlocking process based on the received commands. Here's how the UART interrupt works:

Listing 4.5.2.1: UART Interrupt Setup

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) // interrupt function
{
    if ((flag == 0) && (huart->Instance == USART2)) // make sure the message c
omes from UART2
    {
        if (RX_dat == 'o') // when the board receives 'o'
        {
            BT_Flag = 1;
        }
    }
    HAL_UART_Receive_IT(&huart2, &RX_dat, 1); // enable interrupt again
}
```

In this callback function, the received data is checked, and if it matches the unlock command (`'o'`), the BT_Flag is set to 1.

In the menu.c file, the BT_Flag will be continuously checked to determine if a remote unlock request is made. If BT_Flag is set to 1, the system will trigger the unlocking process.

Listing 4.5.2.2: Unlocking Process

```
int initial_menu(void)
{

      // other menu options

          else if (BT_Flag == 1) { // Bluetooth unlock triggered
        ssd1306_Fill(Black);
        ssd1306_SetCursor(0, 0);
        ssd1306_WriteString("Unlock!", Font_7x10, White);
        ssd1306_UpdateScreen();
```

```
        // …

    }

    return 0; // Stay in the initial menu
}
```

To sum up, the remote unlocking function was achieved by using HC-05 module to receive and pass through the information to STM32. The unlocking message will be sent from the terminals, then the HC-05 will be able to receive the unlocking message and transmit it to STM32. STM32 will then be able to unlock the door.

4.6 Implementation of Fingerprint Unlocking Function

The AS608 fingerprint sensor is an optical biometric sensor designed to capture and authenticate fingerprint patterns. The sensor uses light-based technology to scan the fingerprint and extract a unique set of features, which are then used for comparison during authentication. This verification process relies on several steps that combine optical sensing and advanced algorithms to ensure the security and accuracy of the fingerprint recognition. This section will show how the AS608 Fingerprint Sensor is integrated into the smart door lock system.

4.6.1 AS608 Hardware Setup

The AS608 fingerprint sensor communicates with the STM32 microcontroller through a UART interface. This sensor is capable of capturing, processing, and storing fingerprint templates, which are then used for verification during the authentication process. To set up the AS608 fingerprint sensor with the STM32 microcontroller, connect the following pins:

AS608 communicates through the UART, and USART3 interface is configured on the STM32 for handling the sensor's data. To begin communication, the STM32 must initialize the UART, and AS608 will send and receive data via the UART interface for fingerprint enrolment and verification.

***Table 4.6.1**: Connections Between AS608 and STM32*

AS608 Pins	STM32 Pins
VCC	3.3V
GND	GND
TX	PB11 (UART_3 RX)
RX	PB10 (UART_3 TX)

4.6.2 AS608 Functions Introduction

- Capture Fingerprint Image: The sensor captures an image of the user's fingerprint using the PS_GetImage() function. If the fingerprint is detected, it is stored in the sensor's image buffer.
- Generate Fingerprint Template: The image is processed, and a fingerprint template is generated using the PS_GenChar() function.

- Match Fingerprint: The generated templates are compared to existing ones using the PS_Match() function, ensuring that the user's fingerprint matches the stored template.
- Store Fingerprint Template: Once the template is validated, it is stored in the sensor's memory with the PS_StoreChar() function.
- Fingerprint Scanning: When a user places their finger on the sensor, the PS_GetImage() function captures the fingerprint image.
- Template Generation: The image is processed to generate a fingerprint template using PS_GenChar().
- Template Matching: The generated template is compared against stored templates using the PS_Match() function. If a match is found, access is granted.

4.6.3 Software Implementation of Fingerprint Authentication

In main.c, the USART3 is configured to communicate with the AS608 fingerprint sensor. When a fingerprint scan occurs, the data from the sensor is processed to either enroll or verify the fingerprint:

Listing 4.6.3.1: Function to Record Fingerprint Data

```
extern uint8_t keyPressed;

void Record_FR(void)
{
    uint8_t ensure, step = 0;
    uint8_t retryCount = 0; // Retry count for each step
    char message[32];

    extern uint8_t ID_NUM;
    printf("%d\r\n", ID_NUM);

    while (1)
    {
        ssd1306_Fill(Black);
        switch (step)
        {
        case 0:
            // Prompt user to place finger
          if (keyPressed == 'B'){
            keyPressed = 0;
            return;
          }

            ssd1306_SetCursor(0, 0);
            ssd1306_WriteString("Place Finger", Font_7x10, White);
            ssd1306_SetCursor(0, 20);
            ssd1306_WriteString("Hold B to exit", Font_7x10, White);
            ssd1306_UpdateScreen();

            ensure = PS_GetImage();
```

```
            if (ensure == 0x00)
            {
                ensure = PS_GenChar(CharBuffer1); // Generate feature from fin
gerprint
                if (ensure == 0x00)
                {
                    retryCount = 0;
                    step = 1; // Move to the next step
                }
                else
                {
                    ShowErrMessage(ensure);
                }
            }
            break;

        case 1:
            // Prompt user to place finger again
            ssd1306_SetCursor(0, 0);
            ssd1306_WriteString("Put away", Font_7x10, White);
            ssd1306_UpdateScreen();
            HAL_Delay(1500);
          ssd1306_SetCursor(0, 0);
            ssd1306_WriteString("Place Again", Font_7x10, White);
            ssd1306_UpdateScreen();
            HAL_Delay(1500);

            ensure = PS_GetImage();
            if (ensure == 0x00)
            {
                ensure = PS_GenChar(CharBuffer2); // Generate second feature
                if (ensure == 0x00)
                {
                    retryCount = 0;
                    step = 2; // Move to the next step
                }
                else
                {
                    ShowErrMessage(ensure);
                }
            }
            else
            {
                ShowErrMessage(ensure);
            }
            break;

        case 2:
```

```
            // Match the two features
            ssd1306_SetCursor(0, 0);
            ssd1306_WriteString("Matching", Font_7x10, White);
            ssd1306_UpdateScreen();
            HAL_Delay(250);

            ensure = PS_Match();
            if (ensure == 0x00)
            {
                step = 3; // Move to the next step
            }
            else
            {
                ShowErrMessage(ensure);
                step = 0; // Restart the process if match fails
            }
            break;

        case 3:
            // Register the fingerprint template
            ssd1306_SetCursor(0, 0);
            ssd1306_WriteString("Registering", Font_7x10, White);
            ssd1306_UpdateScreen();
            HAL_Delay(250);

            ensure = PS_RegModel();
            if (ensure == 0x00)
            {
                step = 4; // Move to the storage step
            }
            else
            {
                ShowErrMessage(ensure);
                step = 0; // Restart the process if registration fails
            }
            break;

        case 4:
            // Store the fingerprint template
            ssd1306_SetCursor(0, 0);
            ssd1306_WriteString("Storing ID:", Font_7x10, White);
            sprintf(message, "ID: %d", ID_NUM);
            ssd1306_SetCursor(0, 20);
            ssd1306_WriteString(message, Font_7x10, White);
            ssd1306_UpdateScreen();
            HAL_Delay(1500);

            ensure = PS_StoreChar(CharBuffer2, ID_NUM);
```

```
            if (ensure == 0x00)
            {
                  return; // Successfully recorded and return to menu
            }
            else
            {
                ShowErrMessage(ensure);
                step = 0; // Restart the process if storage fails
            }
            break;
        }

        // Retry handling
        retryCount++;
        if (retryCount >= 5)
        {
            ssd1306_Fill(Black);
            ssd1306_SetCursor(0, 0);
            ssd1306_WriteString("Timeout!", Font_7x10, White);
            ssd1306_UpdateScreen();
            HAL_Delay(1500);
            break; // Return to menu after timeout
        }

        HAL_Delay(500);
    }
}
```

The Record_FR function is responsible for capturing a user's fingerprint and storing it in the fingerprint database. This process occurs in several steps:

- Step 0: Initial Finger Placement The user is prompted to place their finger on the fingerprint sensor. If the fingerprint is successfully captured, the system generates a feature set from the fingerprint image and stores it in the first buffer (CharBuffer1). If there is an error during image capture or feature generation, an error message is displayed.
- Step 1: Second Finger Placement After the first feature is stored, the user is asked to remove and place their finger again. A second feature is generated and stored in the second buffer (CharBuffer2). The system then moves on to matching these two features.
- Step 2: Feature Matching The two captured fingerprint features are compared. If the features match, the system proceeds to the next step. If they do not match, the process is restarted to ensure a successful match.
- Step 3: Fingerprint Registration Once the features match, the fingerprint template is registered in the system. The system attempts to store the fingerprint data. If the registration is successful, the process concludes, and the fingerprint data is saved with an associated ID number.

- Step 4: Storing the Fingerprint Finally, the fingerprint template is stored in the database with an ID number. If the storage is successful, the function ends, and the fingerprint is now available for future verification.

Throughout the process, the system handles retries and timeouts, ensuring that the user is given enough time to provide a good quality fingerprint image while also protecting against too many failed attempts.

Listing 4.6.3.2: Function to Verify Fingerprint Data

```
int verify_FR(void)
{
  SearchResult seach;
  uint8_t ensure;
  char message[20];

  while(1)
  {
    //key_num = KEY_Scan(0);
    ensure = PS_GetImage();
    if(ensure == 0x00) // Successfully get the image
    {
      ensure = PS_GenChar(CharBuffer1);
      if(ensure == 0x00) // Successfully generating information
      {
        ensure = PS_HighSpeedSearch(CharBuffer1, 0, 99, &seach);
        if(ensure == 0x00) //Successfully Searched
        {
          ssd1306_Fill(Black);
          ssd1306_SetCursor(0, 0);
          ssd1306_WriteString("Unlock!", Font_7x10, White); // Unlock
Successfully

          sprintf(message, "ID: %d", seach.pageID); // Convert ID to String
          ssd1306_SetCursor(0, 20);
          ssd1306_WriteString(message, Font_7x10, White); // Show fingerprint
ID

          ssd1306_UpdateScreen();
          HAL_Delay(2000);
          return 1;
        }
        else
        {
          ssd1306_Fill(Black);
          ssd1306_SetCursor(0, 0);
          ssd1306_WriteString("Verification", Font_7x10, White);
          ssd1306_SetCursor(0, 11);
          ssd1306_WriteString("Failed!", Font_7x10, White);
```

```
            ssd1306_UpdateScreen();
            HAL_Delay(2000); // show for 2 seconds
            return 0;
          }
        }
      }
    }
}
```

The verify_FR function is used to authenticate a user by verifying their fingerprint against stored templates. Here's how the function works:

1. **Capture Fingerprint Image** When the user places their finger on the sensor, the system captures the fingerprint image using the PS_GetImage() function. If the image is successfully captured, the function proceeds to generate a feature from the fingerprint using the PS_GenChar() function.
2. **Search for Fingerprint Match** Once the feature is generated, the system searches for a match within the stored fingerprint database using the PS_HighSpeedSearch() function. The search checks the database for the closest matching fingerprint.
3. **Successful Verification** If a match is found, the system displays a success message on the screen and indicates the matching fingerprint's ID. The verification process concludes, and access is granted.
4. **Failed Verification** If no match is found, the system displays a failure message, and the process returns to the beginning, prompting the user to try again.

This process ensures that only registered fingerprints can gain access.

Listing 4.6.3.3: Function to Delete Fingerprint Data

```
void Del_FR(uint16_t id) {
    uint8_t ensure;
    ssd1306_Fill(Black);
    ssd1306_SetCursor(0, 0);
    ssd1306_WriteString("Deleting ID:", Font_7x10, White);

    char message[20];
    sprintf(message, "ID: %d", id);
    ssd1306_SetCursor(0, 20);
    ssd1306_WriteString(message, Font_7x10, White);
    ssd1306_UpdateScreen();
    HAL_Delay(2000);

    ensure = PS_DeletChar(id, 1);
    if (ensure == 0x00) {
        ssd1306_Fill(Black);
        ssd1306_SetCursor(0, 20);
        ssd1306_WriteString("Deleted", Font_7x10, White);
        ssd1306_SetCursor(0, 40);
```

```
        ssd1306_WriteString("Successfully", Font_7x10, White);
    } else {
        ShowErrMessage(ensure);
    }
    ssd1306_UpdateScreen();
    HAL_Delay(2000);
}
```

The **Del_FR** function allows the deletion of a fingerprint from the database using a specific fingerprint ID. Here's how the function works:

- **Prompt User for ID** The user is prompted to enter the ID of the fingerprint they wish to delete. Once the ID is received, the system proceeds to delete the fingerprint associated with that ID.
- **Delete Fingerprint** The system calls the PS_DeletChar() function to remove the fingerprint template from the database. If the deletion is successful, a success message is displayed. If there is an error, an error message is shown to the user.
- **Confirmation** After the deletion, the system displays a confirmation message indicating whether the fingerprint was successfully deleted or if an error occurred.

This function provides the ability to manage the fingerprint database, ensuring that outdated or incorrect data can be removed as needed.

At this stage, all the key functions involved in fingerprint verification have been explained. The STM32 will utilize these functions within the menu, allowing the user to choose different options to achieve various objectives.

4.7 Implementation of Facial Recognition Function

A key component of the door lock system is the Maix Bit K210 development board, which is used for facial recognition tasks. The Maix Bit K210 is a low-cost AI development board powered by a dual-core 64-bit RISC-V processor. It features specialized hardware accelerators that optimize AI workloads, making it ideal for real-time image processing and biometric authentication. In addition, the Maix Bit Kit will also include a camera to capture real time image, and a full colour OLED screen to display the image.

The Maix Bit K210 utilizes its KPU (Kendryte Processing Unit) to efficiently run machine learning models, such as YOLO (You Only Look Once) for face detection, and face recognition models for feature extraction. The KPU enables high-performance processing of these tasks locally, reducing latency and reliance on external servers. This allows the system to detect and recognize faces in real-time, providing an additional layer of security for the smart door lock system.

4.7.1 Maix Bit Hardware Setup

The Maix Bit K210 development board communicates with the STM32 microcontroller through a UART or I2C interface, depending on the configuration chosen for the system. This board is responsible for processing facial recognition tasks using deep learning models and real-time image processing. The Maix Bit K210 captures images through a connected camera module and performs face detection, feature extraction, and recognition. To set up the Maix Bit K210 with the STM32 microcontroller, connect the following pins:

Table 4.7: Connections Between Maix Bit and STM32

Maix Bit Development Board Pins	STM32 Pins
VCC	5V
GND	GND
TX	PA10 (UART_1 RX)
RX	PA9 (UART_1 TX)

4.7.2 Maix Bit Software Implementation

The Maix Bit K210 processes real-time images captured by the camera for facial recognition. It compares the detected face with stored face data and calculates a similarity score. If the score exceeds the predefined threshold (set to 85 in this system), the K210 recognizes the face as valid. It then sends the result via UART to the STM32 microcontroller, which processes the authentication outcome. Due to space constraints, only the key components of the system will be explained in detail below:

Listing 4.7.2.1: Functions about Timer Interrupt Callback

```
# Timer interrupt callback function
tem = ''
b = []
check = 0  # Store
save = 0  # Temporary storage
switch = 0  # Switch
delete = 0

def on_timer(timer):  # Callback function
    global check
    global save
    global delete
    data = []
    data = com.read(2)
    if data != None:
        print(data)
        if data == b'A':
            check = 1  # Represent storing face features
            save = 0  # Do not save to SD card
        elif data == b'C':
            check = 1  # Represent storing face features
            save = 1  # Save to SD card
        elif data == b'D':  # If receive delete all command
            # delete = 1  # Call function to delete all face features
            delete_all_faces()

# Timer interrupt initialization
tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_ONE_SHOT,
period=500,
             unit=Timer.UNIT_MS, callback=on_timer, arg=on_timer, start=False)
```

This code defines a timer interrupt callback function that handles UART communication and triggers certain actions based on the received commands.

The on_timer function is called every 500 milliseconds when the timer interrupt occurs. Inside this function, the program reads 2 bytes of data from the UART communication buffer using com.read(2). If valid data is received, it processes the data to perform different actions:

- If the received data is b'A', it sets the check flag to 1, indicating that the system should store facial features. The save flag is set to 0, meaning the features will not be saved to the SD card.
- If the received data is b'C', it sets the check flag to 1 again (for storing facial features), but this time it sets the save flag to 1, indicating that the features should be saved to the SD card.
- If the received data is b'D', it triggers the deletion of all stored face features by calling the delete_all_faces() function. The deletion flag (delete) is not used in this snippet but may be relevant for other logic in the system.

The timer itself is initialized with the following configuration: it operates in one-shot mode, meaning it will trigger only once every 500 milliseconds. The timer starts when explicitly instructed to do so in the main program. The on_timer function is set as the callback, which means it will execute every time the timer interrupt is triggered.

Listing 4.7.2.1: Functions about Timer Interrupt Callback

```
anchor = (1.889, 2.5245, 2.9465, 3.94056, 3.99987, 5.3658, 5.155437, 6.92275,
6.718375, 9.01025)  # Anchor for face detection
dst_point = [(44,59),(84,59),(64,82),(47,105),(81,105)]  # Standard face key
point positions
a = kpu.init_yolo2(task_fd, 0.5, 0.3, 5, anchor)  # Initialize face detection
model
img_lcd = image.Image()  # Set display buffer
img_face = image.Image(size=(128,128))  # Set 128*128 face image buffer
a = img_face.pix_to_ai()  # Convert image to KPU accepted format

while(1):  # Main loop
    check = 0
    save = 0
    delete = 0
    tim.start()  # Start timer interrupt
    img = sensor.snapshot()  # Capture an image from the camera
    clock.tick()  # Record time for frame rate calculation
    code = kpu.run_yolo2(task_fd, img)  # Run face detection model to get face
bounding box coordinates
    # b = img.draw_string(0, 0, ("tem"), color=(0, 255, 0), scale=2)
    if code:  # If face is detected
        for i in code:  # Iterate through bounding box coordinates
            # Cut face and resize to 128x128
            a = img.draw_rectangle(i.rect())  # Draw face bounding box on
screen
```

```
            face_cut = img.cut(i.x(), i.y(), i.w(), i.h())  # Crop face from
image
            face_cut_128 = face_cut.resize(128, 128)  # Resize cropped face to
128x128
            a = face_cut_128.pix_to_ai()  # Convert cropped face to KPU format
            # Landmark for face 5 points
            fmap = kpu.forward(task_ld, face_cut_128)  # Run face landmark
detection model
            plist = fmap[:]  # Get landmark points
            le = (i.x() + int(plist[0] * i.w() - 10), i.y() + int(plist[1] *
i.h()))  # Calculate left eye position
            re = (i.x() + int(plist[2] * i.w()), i.y() + int(plist[3] *
i.h()))  # Calculate right eye position
            nose = (i.x() + int(plist[4] * i.w()), i.y() + int(plist[5] *
i.h()))  # Calculate nose position
            lm = (i.x() + int(plist[6] * i.w()), i.y() + int(plist[7] *
i.h()))  # Calculate left mouth corner position
            rm = (i.x() + int(plist[8] * i.w()), i.y() + int(plist[9] *
i.h()))  # Calculate right mouth corner position
            a = img.draw_circle(le[0], le[1], 4)
            a = img.draw_circle(re[0], re[1], 4)
            a = img.draw_circle(nose[0], nose[1], 4)
            a = img.draw_circle(lm[0], lm[1], 4)
            a = img.draw_circle(rm[0], rm[1], 4)  # Draw circles on the
respective key points
            # Align face to standard position
            src_point = [le, re, nose, lm, rm]  # Face key points from the
image
            T = image.get_affine_transform(src_point, dst_point)  # Get affine
transform matrix for face alignment
            a = image.warp_affine_ai(img, img_face, T)  # Apply affine
transformation to align face to standard
            a = img_face.ai_to_pix()  # Convert aligned face to KPU format
            del(face_cut_128)  # Release cropped face image
            # Calculate face feature vector
            fmap = kpu.forward(task_fe, img_face)  # Calculate 196-dimensional
feature vector of aligned face
            feature = kpu.face_encode(fmap[:])  # Get feature encoding
            reg_flag = False
            scores = []  # List to store feature comparison scores
            for j in range(len(record_ftrs)):  # Iterate through stored
features
                score = kpu.face_compare(record_ftrs[j], feature)  # Compare
current face feature with stored features
                scores.append(score)  # Add score to list
            max_score = 0
            index = 0
```

```
            for k in range(len(scores)):  # Iterate through all comparison
scores, find the maximum score and index
                if max_score < scores[k]:
                    max_score = scores[k]
                    index = k
            # if delete == 1:
            # delete_all_faces()
            # delete = 0
            if max_score > 80:  # If maximum score is greater than 80, it can
be recognized as the same person
                a = img.draw_string(i.x(), i.y(), ("%s :%2.1f" %
(names[index], max_score)), color=(0, 255, 0), scale=2)  # Display name and
score
                com.write('B')
            else:
                a = img.draw_string(i.x(), i.y(), ("X :%2.1f" % (max_score)),
color=(255, 0, 0), scale=2)  # Display unknown and score
                if check == 1:  # If data is received from UART
                    check = 0
                    record_ftr = feature
                    record_ftrs.append(record_ftr)  # Add current feature to
known feature list
                    if save == 1:  # If saving
                        save = 0
                        a = img.draw_string(100, 100, "Stor successful",
color=(0, 255, 0), scale=2)
                        print("Stor successful")
                        save_feature(record_ftr)  # Save to SD card
                    break
    a = img.draw_string(0, 220, "face", color=(255, 0, 0), scale=2)  # Display
unknown and score
    a = lcd.display(img)  # Refresh screen display
```

1. **Face Detection and Initialization:**

The code starts by defining the anchor points for face detection using the anchor variable and the standard face key point positions (dst_point). These are crucial for aligning detected faces with the model's expected configuration. The YOLOv2 model for face detection is initialized using the kpu.init_yolo2() function. This model is responsible for detecting faces in images captured by the camera. Buffers for the face image and the camera display are set up using img_lcd and img_face.

2. **Main Loop:**

Inside the main loop, the camera captures an image using sensor.snapshot(). This image is then passed through the face detection model using kpu.run_yolo2(), which returns the bounding box coordinates of any detected faces in the image. If faces are detected (i.e., code is not empty), the program iterates through each face's bounding box coordinates.

3. **Face Cropping and Resizing:**

For each detected face, a bounding box is drawn on the image (img.draw_rectangle(i.rect())), and the face region is cropped from the image using img.cut(i.x(), i.y(), i.w(), i.h()). The cropped face is then resized to 128x128 pixels using face_cut.resize(128, 128) for better processing by the facial recognition model.

4. **Face Landmark Detection:**

The resized face is passed through the face landmark detection model (task_ld) using kpu.forward(task_ld, face_cut_128), which detects key points on the face (such as the eyes, nose, and mouth). The landmark positions are extracted from the model output and used to compute the positions of the eyes, nose, and mouth corners.

5. **Face Alignment:**

The detected facial landmarks (le, re, nose, lm, rm) are used to align the face to a standard position. This is done using an affine transformation (image.get_affine_transform()) based on the landmark points (src_point), which maps them to predefined standard positions (dst_point). The aligned face is then transformed and converted into the format suitable for KPU processing (a = image.warp_affine_ai(img, img_face, T)).

6. **Feature Extraction:**

The aligned face is passed through the feature extraction model (task_fe) using kpu.forward(task_fe, img_face) to extract a 196-dimensional feature vector that represents the face. The feature encoding is obtained from kpu.face_encode(fmap[:]), which produces a unique numerical representation of the face for comparison.

7. **Face Comparison:**

The extracted feature vector is compared to the previously stored face features in record_ftrs using kpu.face_compare(). This function compares the current face feature with each stored face feature and returns a similarity score. The code iterates through the scores, finds the highest one, and checks if it is above the threshold (e.g., 80). If the score exceeds the threshold, the face is recognized as a match, and the corresponding name from the names list is displayed along with the score.

8. **Storing and Saving Features:**

If the face is not recognized, the system allows the user to save the current face feature to the SD card for future recognition. If the check flag is set to 1 and save is set to 1, the face features are added to record_ftrs and saved to the SD card using the save_feature() function.

9. **Display Updates:**

The system continuously updates the LCD display to show the current status. If the face is recognized, the name and score are shown on the screen; if the face is unknown, "X" and the score are displayed instead.

4.7.3 STM32 Software Implementation

The logic used by the STM32 to process the facial recognition results is almost identical to the method it uses for handling Bluetooth unlocking. For more details, please refer to section 5.5.2 of this document.

Listing 4.7.3.1: UART Interrupt Setup

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) // interrupt function
{
    if ((flag == 13) && (huart->Instance == USART1)) // make sure the message
comes from UART1
    {
        if (RX_dat == 'B') // when the board receives 'B'
        {
            facial_flag = 1;
        }
        HAL_UART_Receive_IT(&huart1, &RX_dat, 1); // enable interrupt again
    }
    HAL_UART_Receive_IT(&huart2, &RX_dat, 1); // enable interrupt again
}
```

Listing 4.7.3.2: Unlocking Process

```
int menu_facial_check(void)
{

      // ……

    if (facial_flag == 1) { // If facial recognition is successful
        facial_flag = 0; // Reset flag
        ssd1306_Fill(Black);
        ssd1306_SetCursor(0, 0);
        ssd1306_WriteString("Unlock!", Font_7x10, White); // Display unlock
message
        ssd1306_UpdateScreen();

      // ……

    }



    return 13; // Stay in the current menu
}
```

4.8 Implementation of the Power Supply Module

In this system, two MB102 DC-DC power supply modules connected in parallel were used, each powered by a separate **PP3 battery** with a voltage of **9V**. This configuration ensures that the system can meet the power requirements of all components, especially the facial recognition module, which requires a significant amount of current. The MB102 modules provide **3.3V** and **5V** outputs[7], which are essential for powering different parts of the system.

The facial recognition module needs approximately 600mA of current to function[8]. However, a single MB102 module can only supply a maximum of 700mA of current[7]. Given this, a

single module would be insufficient to handle the facial recognition module's power demand along with any other potential components.

4.9 Implementation of the Custom PCB

The PCB used in this project is a two-layer design that integrates various peripherals for the smart door lock system. It connects components like the STM32 microcontroller, fingerprint sensor, keypad, Wi-Fi module, and the K210 development board for facial recognition. The purpose of the PCB is to ensure smooth communication between all these modules, enhancing the system's overall reliability and performance. The detailed design is shown in the next page:

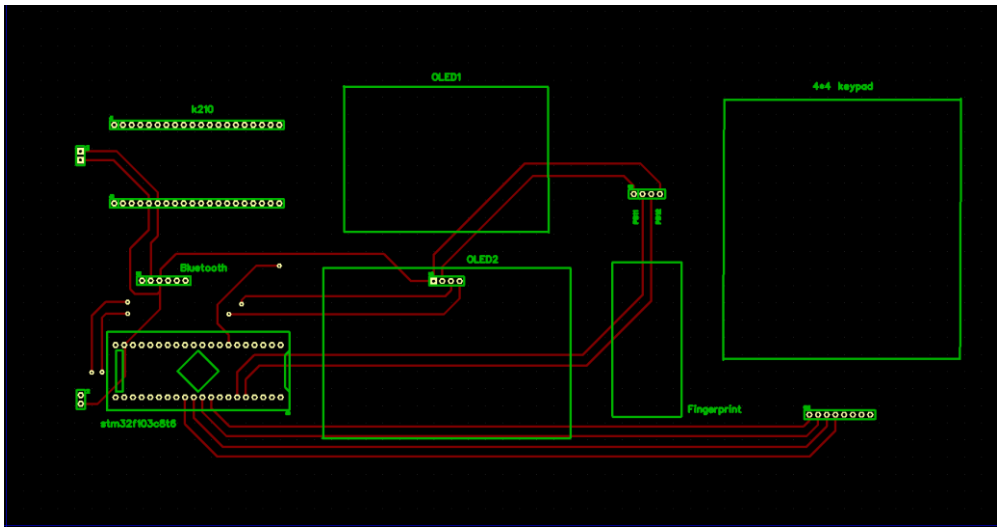


***Figure 4.9.1**: Top Layer Design of the PCB*

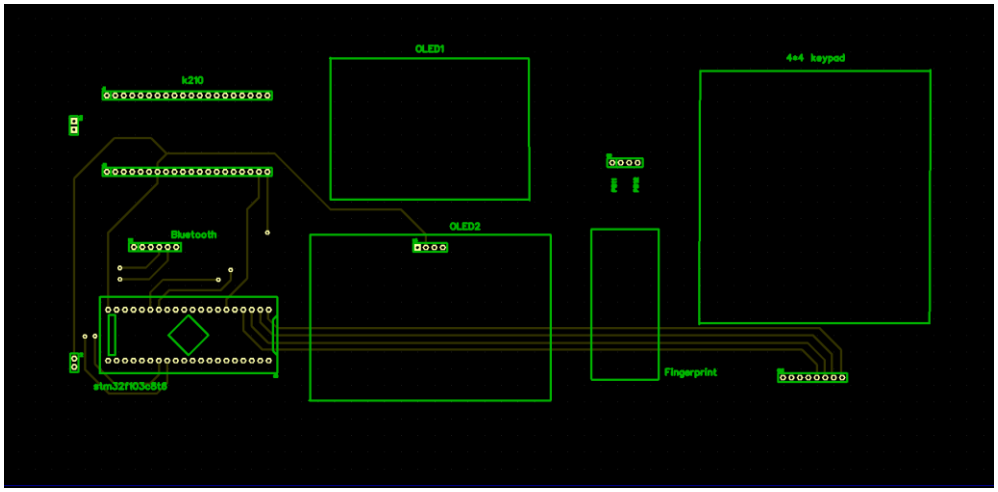


***Figure 4.9.2**: Bottom Layer Design of the PCB*

4.10 Workflow Introduction

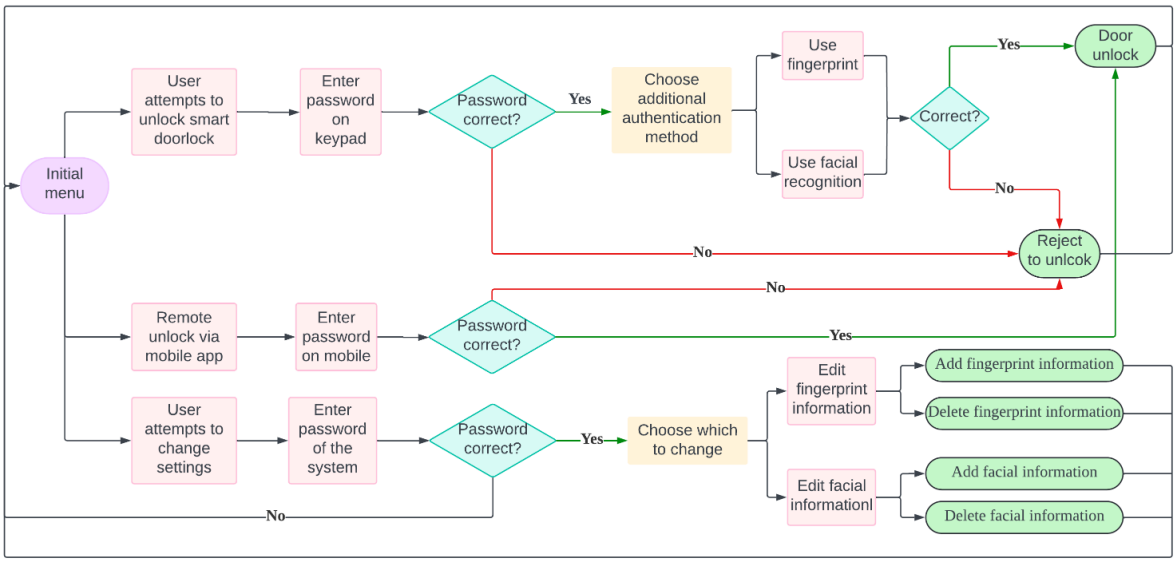


***Figure 4.10**: Workflow Diagram of the System*

Through the diagram shown above, there are three main paths for unlocking the door:

1. Enter password + fingerprint recognition.
2. Enter password + facial recognition.
3. Bluetooth unlocking.

Furthermore, the user can enter a password to access the settings menu, where they can modify (add or delete) biometric data. Additionally, the password to access settings is a six-digit code automatically generated by the system and cannot be modified, whereas the unlocking password is a four-digit code that can be changed within the settings.

5 Results, analysis, and discussion

This section provides the results of the unlocking process, accompanied by screenshots that demonstrate the system's functionality. It then presents an analysis of the system's notable strengths—such as reliability, security, and user-friendliness—along with a discussion of its limitations. By highlighting both achievements and areas for improvement, this section offers a balanced assessment of the system's overall performance and points to opportunities for further development.

5.1 Overview of the Whole System

After all, the final version of the door lock system is shown above.

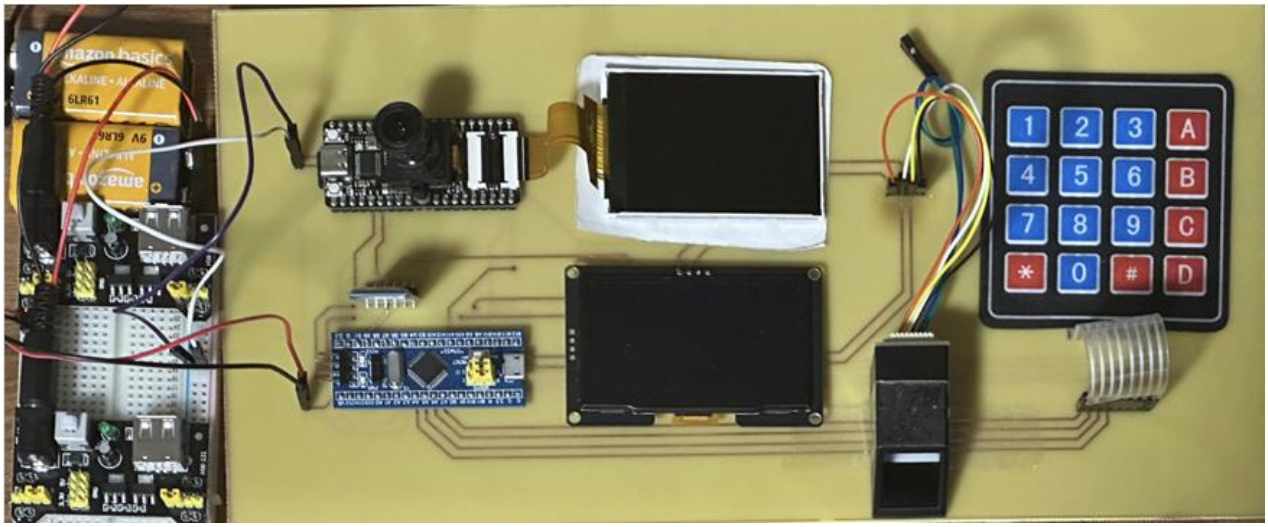


Figure 5.1.1: *Real-time Image of the System*

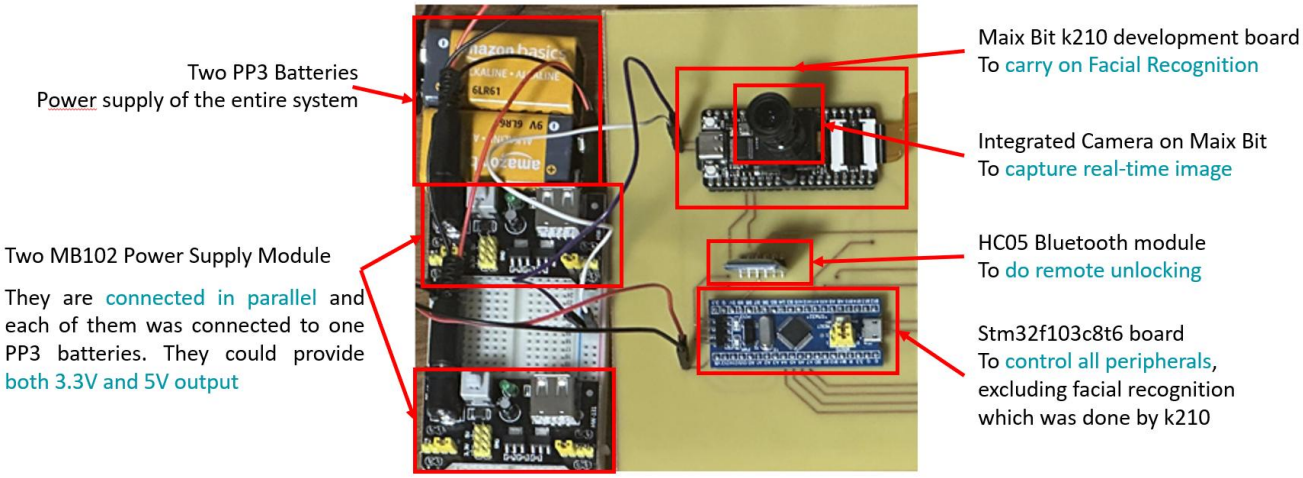


Figure 5.1.2: *Break Down of Components in the System (1)*

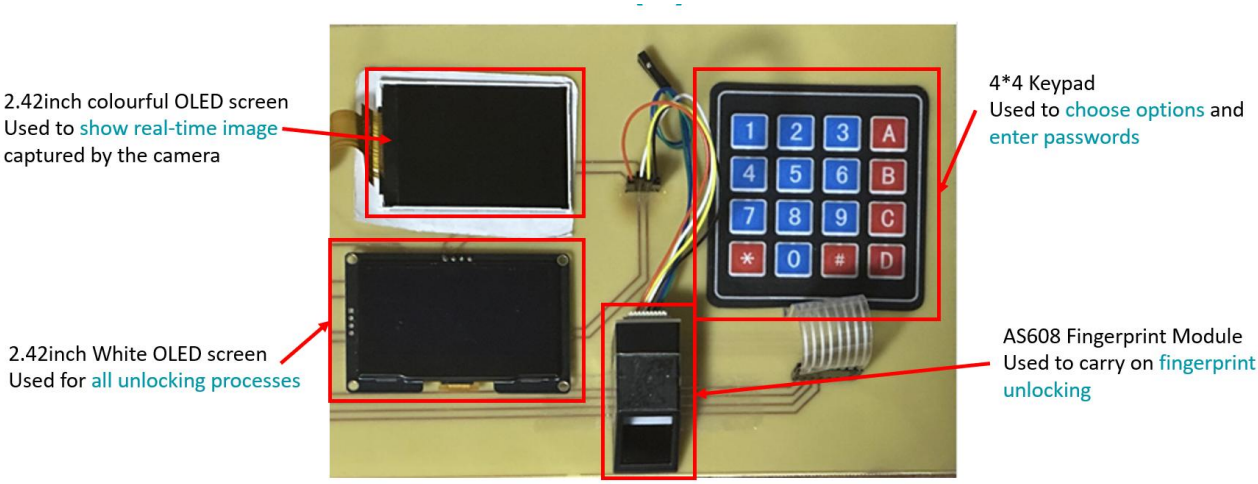


Figure 5.1.3: *Break Down of Components in the System (2)*

5.2 Unlocking through Bluetooth (Remote Unlocking)

To unlock through Bluetooth, download a Bluetooth serial communication application on any phone, find the target:

After that, simply send ‘o’ from the terminal, then the system will unlock.

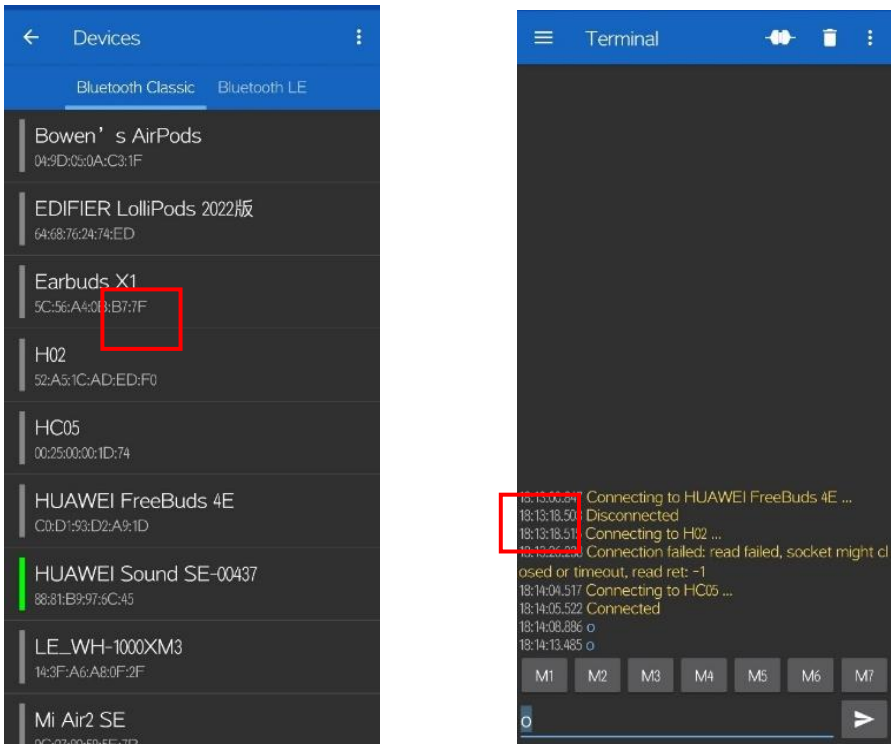
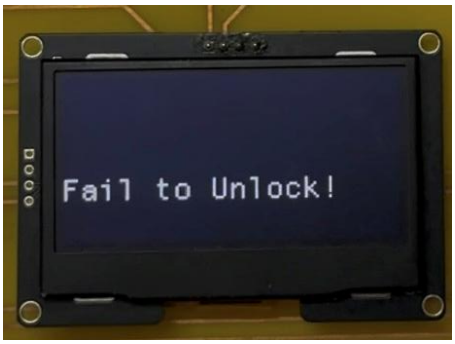


Figure 5.2.1 & 5.2.2*: Screenshots of the Bluetooth Serial Terminal software*

5.3 Unlocking through PIN + Fingerprint Verification

5.3.1 PIN Verification

To verify the PIN, user should input the 4-digit PIN using the 4*4 keypad matrix. If the PIN is correct, the OLED Screen will show: ‘PIN correct!’. If not, it will show ‘Fail to Unlock’



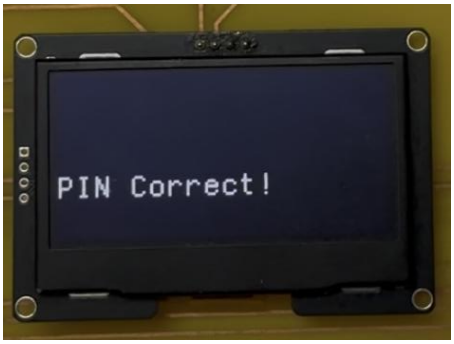


***Figure 5.3.1.1 & 5.3.1.2**: Real-time Images of the OLED screen*

5.3.2 Fingerprint Verification

After successfully verifying the PIN, the menu for choosing different unlocking methods will be shown. In this stage, to use fingerprint verification, simply choose '1. Fingerprint':

***Figure 5.3.2.1**: Real-time Images of the Verification Menu*

Afterwards, the system will ask the user to place the finger on the fingerprint sensor. If the fingerprint data matches any fingerprint data stored into the library, the system will unlock and show the ID of the fingerprint stored.

***Figure 5.3.2.2 & 5.3.2.3**: Real-time Images of process of Fingerprint Verification*

5.3.3 Modifying Fingerprint Data

The user could also access settings to manage the fingerprint data by inputting a 6-digit system PIN. The process to input the PIN is mostly the same as in section 5.3.1. Please refer to that section for detail.

Upon successfully implemented the data, there will also be a menu to choose to modify different types of data. In this case, choose '1. Fingerprint'. After that, the user will need to

input the ID of the fingerprint. This could be done on the keypad and the range of ID is limited from 0 to 9.

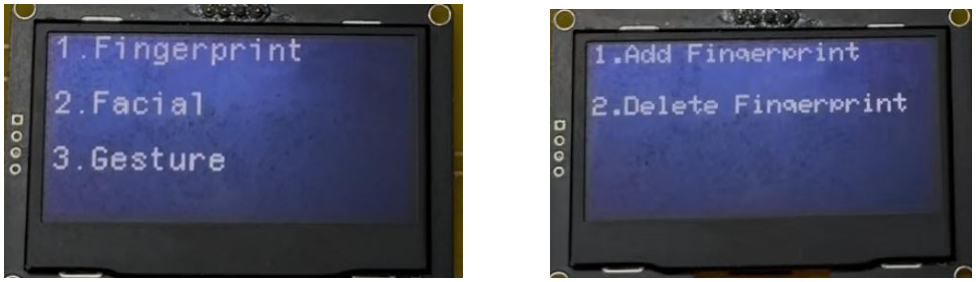


Figure 5.3.3.1*: Real-time Images of the Verification Menu*

Figure 5.3.3.2*: Real-time Images of the Menu to Modify Facial Data*

Then, the user will be needed to place the finger on the sensor twice to make sure the record is reliable:

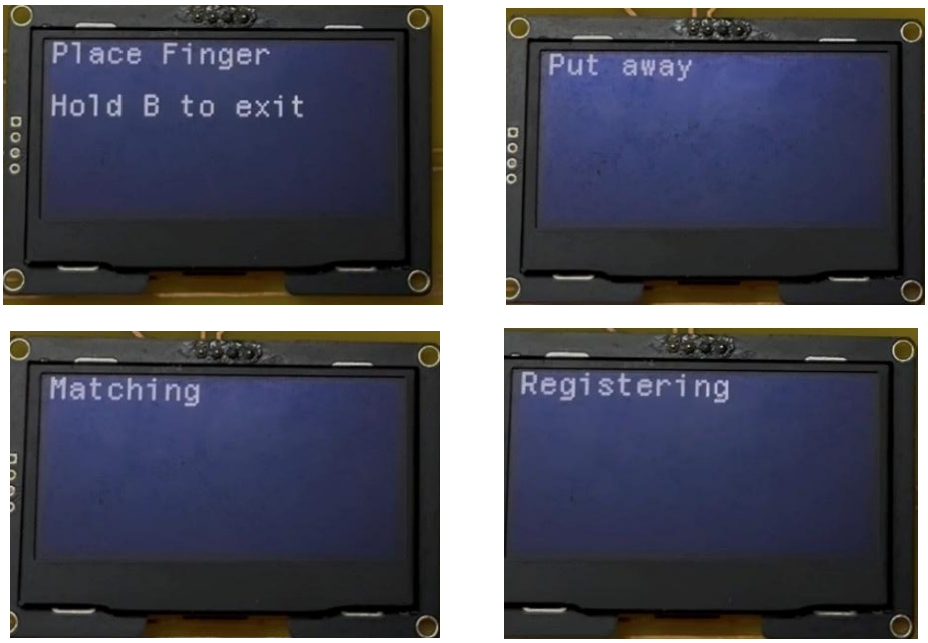


Figure 5.3.3.2 – 5.3.3.5*: Real-time Images of the Recording Process*

Lastly, if the process is successful, the system will show 'Storing ID:', with the ID of the fingerprint given. If not, it will show 'record unsuccessful', with error reason given.

Figure 5.3.3.2 – 5.3.3.5*: Real-time Images of the Recording Result*

Furthermore, to delete certain fingerprint data, referring to figure 5.3.3.2 the user should choose ‘2. Delete Fingerprint’ and input the ID of the fingerprint to delete.

5.4 Unlocking through PIN + Facial Recognition

To access facial recognition function, user should first input a valid PIN. The PIN verification part in this section will be the same as in section 5.3.1. Please refer to 5.3.1 for detail.

5.4.1 Facial Verification

After successfully verifying the PIN, the menu for choosing different unlocking methods will be shown. In this stage, to use facial verification, simply choose ‘2.facial’:

Figure 5.4.1.1*: Real-time Images of the Verification Menu*

Afterwards, the system will ask the user to look at the camera. If a face shows in front of the camera, the OLED screen connected to Maix Bit will show the ID of the face and its score. If the face matches any face stored in the library, the words will turn green and the system will be unlocked.

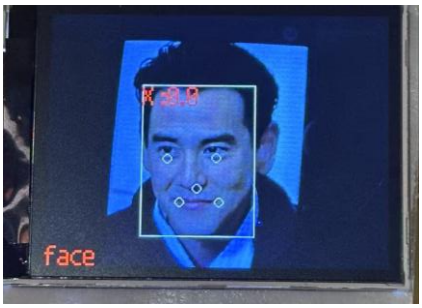


Figure 5.4.1.2*: Real-time Image When a Recorded Face is Shown*

Figure 5.4.1.3*: Real-time Image When a not Recorded Face is Shown*

5.4.2 Modifying Facial Data

The user could also access settings to manage the fingerprint data by inputting a 6-digit system PIN. The process to input the PIN is mostly the same as in section 5.3.1. Please refer to that section for detail.

Upon successfully implemented the data, there will also be a menu to choose to modify different types of data. In this case, choose ‘2. Facial’. After that, the user will need to show the face in front of the camera, and press ‘1. Add Face’ to record the facial data.

Figure 5.4.2.1*: Real-time Images of the Verification Menu*

Figure 5.4.2.2*: Real-time Images of the Menu to Modify Facial Data*

Upon successful recording, the words shown on the OLED connected to Maix Bit will turn from red to green, just like figure 5.4.1.2 and 5.4.1.3. Furthermore, to delete facial data, referring to figure 5.4.2.2 the user should choose '2. Delete All Faces', then all the facial data will be deleted.

5.5 Limitations and Possible Solutions

Although the current Smart Door Lock System demonstrates substantial progress in functionality, there remain certain limitations that warrant attention. These constraints arise from factors such as hardware resource constraints, communication overhead, and the complexity of biometric authentication algorithms. By examining the identified issues, readers gain insight into why the system operates within its present scope and how targeted refinements might lead to future improvements. This section outlines the primary limitations observed so far, highlighting areas that may benefit from optimization or re-design in subsequent iterations.

5.5.1 Limitations in Keypad Matrix

The current implementation relies on software-based debouncing methods to filter out unintended signals caused by switch bounce in the 4×4 keypad matrix. However, during actual input, intermittent bouncing events may still lead to issues such as registering multiple input digits when the user has pressed a key only once, or even detecting incorrect key values. These inaccuracies can disrupt the user experience and cause confusion when entering critical data such as PINs.

A potential solution is to incorporate hardware debouncing circuits—often built with simple RC (resistor-capacitor) filters or Schmitt triggers—to minimize the noise at the physical level before signals reach the microcontroller. This approach typically reduces the occurrence of false triggers more reliably than software debouncing alone, providing a more stable input interface for keypad-based systems.

5.5.2 Limitations in HC-05 Bluetooth Module

Standard Bluetooth technology known as classic Bluetooth serves as the connection method for this HC-05 module. Some smartphones including iPhones do not work with regular Bluetooth technology which prevents them from connecting with this device module. The problem reduces the wireless system's usefulness in places where iOS users need to connect their devices. Replacing the standard Bluetooth with a Low Energy version opens more device compatibility and connects to multiple operating systems easier.

5.5.3 Limitations in Power Supply Module

Using two HC-05 modules increases the size of the power supply unit while wasting power resources. A system with parallel connected HC-05 modules needs substantial size to maintain constant voltage that generates excessive heat and fails to convert energy effectively. Future versions will do better if they include a DC-DC module that makes the power supply smaller and more effective.

6 Conclusions and future work

6.1 Conclusions

The smart door lock system uses one design to provide different ways to unlock: fingerprints, PIN entries, facial recognition, and remote Bluetooth access at a reasonable price. To meet performance needs while securing biometric information the project uses both an STM32 microcontroller and K210 development board. Each security layer provided by the fingerprint and facial recognition modules makes unauthorized entry more difficult for users.

This project successfully creates a low-cost expandable device that meets its original security requirements. The demonstration proves that many authentication methods can work together effectively through a unified embedded platform. The system provides opportunities for future security improvements that maintain user convenience with reliable hardware.

6.2 Future Work

Our next step involves developing the smart door lock system by redesigning hardware elements while adding support for more devices and making the system easier to use. Potential areas of enhancement include:

1. **Hardware Debouncing**

   Using debouncing hardware combining with software would help the keypad matrix make fewer incorrect readings than just a system update. The hardware upgrade will make it easier for users while protecting them from wrong PIN entries.

2. **Upgrade to BLE Modules**

   The device can connect to iOS systems and last longer on battery power when it switches from an HC-05 Bluetooth module to BLE. This change lets more users access the system with no interruptions.

3. **Optimized Power Supply**

   Building dedicated power systems helps improve total efficiency by lowering waste heat production regardless of what detection methods the system activates. Safer long-term operations of the device come from a smaller power circuit design.

4. **Enhanced Facial Recognition Algorithms**

   Placing better and more powerful face-detection models on the K210 development board helps detect faces accurately under all lighting situations. Integrating authenticating features like clicking movements detects alive humans to boost security.

5. **User Interface and Feature Expansion**

   The menus and display settings should be made more readable through simplification and multilingual options. The system can serve practical security goals better when it records failed entry attempts and sends alerts to smartphone devices.

6. **Full Mechanical Integration**

   Collaborating with mechanical engineers to design a dedicated lock enclosure and integrating the electronics with a physical locking mechanism would transform the system

into a fully functional door lock. This housing would incorporate the motor or solenoid for physical locking and unlocking, while ensuring tamper resistance, ease of installation, and a more professional appearance suitable for commercial production.

The smart door lock system will grow into future security basics through these improvements while remaining easy for users to operate.

7 References

[1] Lwin, H., Khaing, A., Tun, H."Automic door access system using face recognition", International Journal Of Scientific & Technology Research, July 2015.

[2] G.Senthikumar et al., "Embedded Image Capturing System Using Raspberry Pi System", International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), April 2014.

[3] M. Shanthini, G. Vidya and R. Arun, "IoT Enhanced Smart Door Locking System," *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, Tirunelveli, India, 2020, pp. 92-96

[4] D. Han, H. Kim and J. Jang, "Blockchain based smart door lock system," *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South), 2017, pp. 1165-1167, doi: 10.1109/ICTC.2017.8190886.

[5] M. Waseem, S. A. Khowaja, R. K. Ayyasamy and F. Bashir, "Face Recognition for Smart Door Lock System using Hierarchical Network," *2020 International Conference on Computational Intelligence (ICCI)*, Bandar Seri Iskandar, Malaysia, 2020, pp. 51-56, doi: 10.1109/ICCI51257.2020.9247836.

[6] Afiskon, *stm32-ssd1306*[Source code]. Available: https://github.com/afiskon/stm32-ssd1306. Accessed: Apr. 19, 2024.

[7] Handson Technology, MB102 Breadboard 3.3V/5V Power Supply Data Sheet, handsontech.com, https://www.handsontec.com/dataspecs/mb102-ps.pdf.

[8] Sipeed Technology, Sipeed-Maix-Bit specification, wiki.sipeed.com, https://wiki.sipeed.com/soft/maixpy/en/develop_kit_board/maix_bit.html

8 Appendices