SECURITY ACCESS CONTROL SYSTEM

BY
Sriya Garde and Krishna Suhagiya

Final Project Report
ECEN 5613 Embedded Systems Design
December 16th, 2023

Table of Contents

Table of Figures

ACRONYMS

RFID	Radio Frequency Identification
NFC	Near Field Communication
OLED	Organic Light Emitting Diode
SPI	Serial Programming Interface
I2C	Inter-Integrated Circuit
UID	Unique Identifier

1 Introduction

Inspired by the modern-day approach to enforce and enhance security using modern technologies like RFID, Barcode, Wireless and QR Code, we have developed a prototype of a Security Access control system. This system will recognize the user cards as Valid and Invalid, and accordingly will provide the respective access flow.

1.1 System Overview

The system is designed using STM32F411E-DISCO microcontroller.

For our System, we have used RFID technology – MFRC522 Module with NFC tags, Key cards and Buff One cards as validators.

Our main inputs consist of the RFID cards - valid and invalid both and a keypad through which, we take inputs in the form of passwords. If the key card does not get validated, we use a security password to provide one time access to the system or an admin password to add the card as a valid card to the system.

For Outputs, we are using OLED to display the status of the validation at each point of time, and we are also employing audio outputs via buzzer and voice recording and playback module with a speaker to make the system more inclusive.

1.2 Motivation for the Project

When we were researching ideas for our project, our interests converged around the idea of exploring how an efficient Security system would work.

We were intrigued by how large institutions and Industries employ access control with a simple, wireless card reading and detection system.

Our main motivation behind doing this project was to explore and work more with a system like the Buff one card system that we currently have in the university. We wanted to learn more about the access control that has been employed in such an environment with the help of access cards and authenticators and design a system with a similar functionality.

2 Technical Description

This chapter details the technical information about the security access control system developed.

2.1 Block Diagram

The block diagram of our system is depicted in figure **Figure 1**.

- At the center is the Controller we are using to design the system – STM32F411E-DISCO Board.
- The inputs to the controller come from the RFID Reader block and the Keypad.
- The controller drives the Outputs for Voice Recorder Module – Speaker output, the Buzzer and the OLED Module.
- The controller processes the inputs of NFC Cards by reading the UID of these cards via the RFID reader, the inputs of passcodes entered via the Keypad and provides appropriate outputs via messages on the OLED display, the beeping of the Buzzer and the playback of recorded messages through the Playback module and the Speaker.

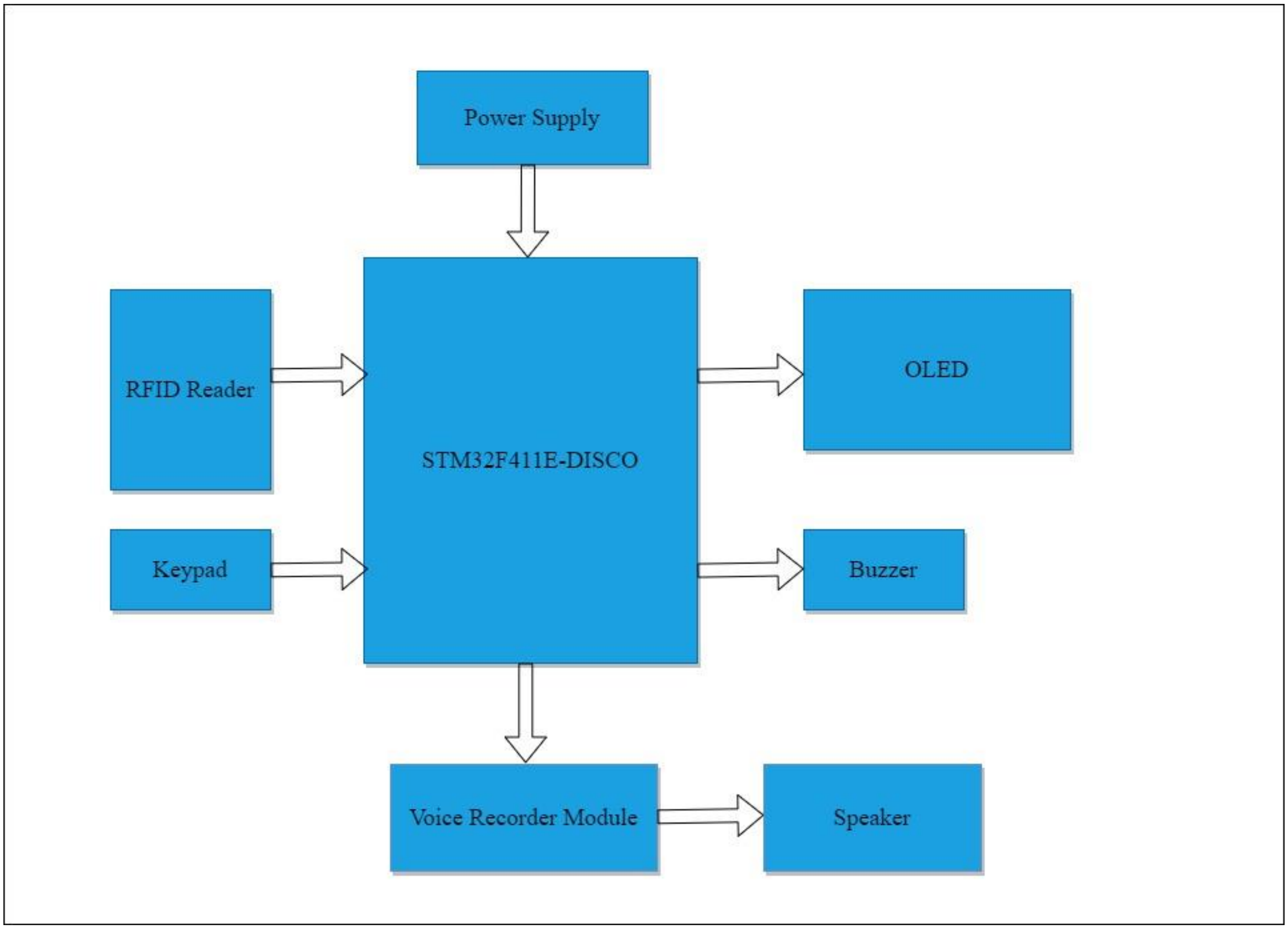


Figure 1: Security Access Control System Block Diagram

We were initially planning on using AT89C51 as our controller for the system but considering the high value of ARM processors in the embedded systems domain, we decided that STM32 would be a better fit for our application. This choice also gave us a chance to work with STM32 drivers for I2C, SPI, Clocks, Timers, and processing GPIO input-outputs.

2.2 Components Used

This section describes the various components of the system.

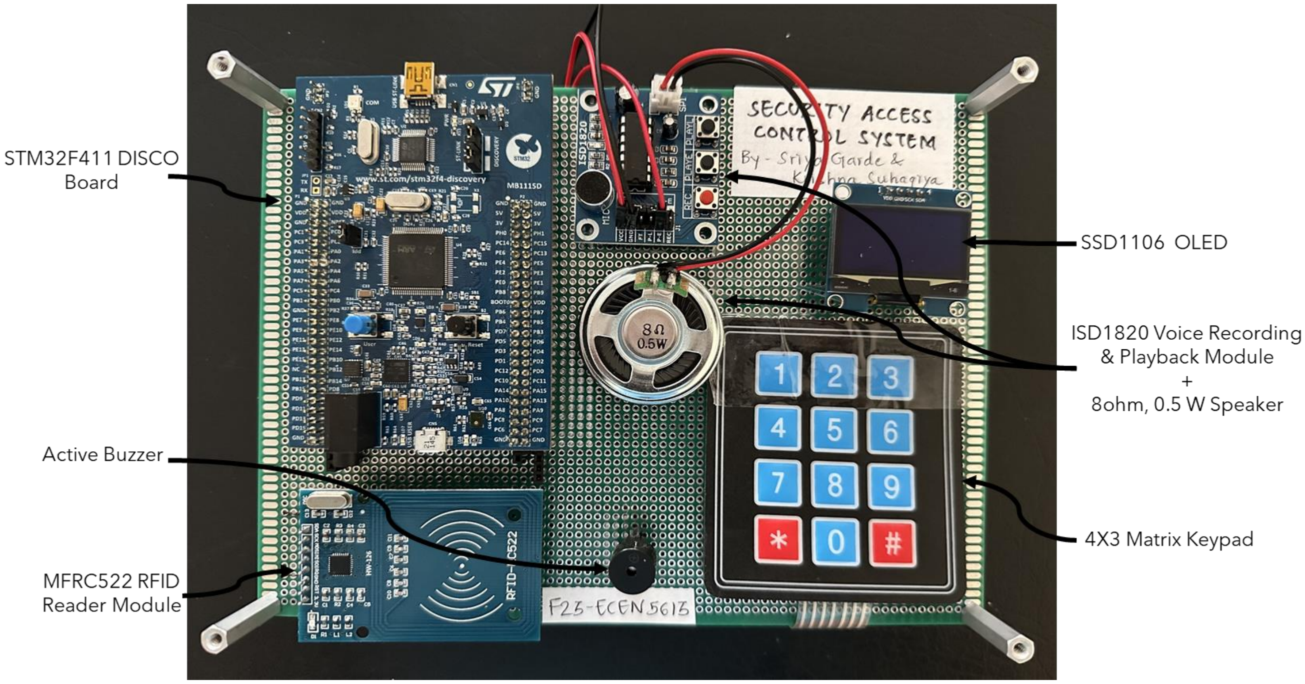

STM32F411 DISCO Board
SSD1106 OLED
ISD1820 Voice Recording & Playback Module + 8ohm, 0.5 W Speaker
Active Buzzer
4X3 Matrix Keypad
MFRC522 RFID Reader Module
SECURITY ACCESS CONTROL SYSTEM
By - Sriya Garde & Krishna Suhagiya
F23-ECEN5613

Figure 2: Security Access Control System PCB

- **STM32F411E-DISCO board**

The STM32F411E-DISCO was chosen as a target board to have more exposure and learning about coding with ARM controller and by working with the SPI and I2C communication protocols and interfacing the modules with it.

- **RFID RC522 Module**

To learn how the RFID system works, RFID RC522 module was chosen with its compatible NFC cards.
The RC522 RFID reader module communicates with RFID tags by creating a 13.56 MHz electromagnetic field. It can read and write RFID cards and tags built using ISO/IEC 14443 protocol.
The RC522 can communicate with a microcontroller over a 4-pin SPI with a maximum data rate of 10 Mbps. It also supports communication over I2C and UART protocols. We opted for SPI communication protocol as it is fast compared to I2C.
The RC522's internal transmitter can drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry.
RC522 RFID reader works at the read range up to 1m and the NFC tags with working read/write range upto 10cm. We can read the card up to ~5cm in our system.

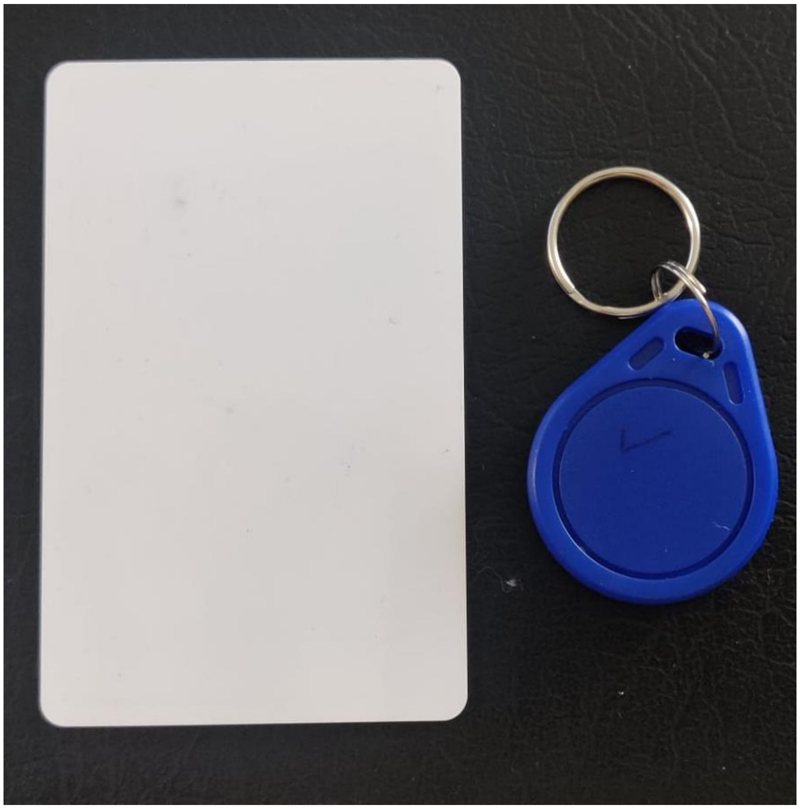


Figure 3: NFC card and keycard

The hardware connections of the RFID RC522 module with the STM32 board are mentioned in 2.4.2
Input DC voltage range: 2.5-3.3 volts.

- **RFID Tags**

We have used 3 types of RFID Cards to validate the functioning of our system.

1. IC Card – The white RFID card set as Valid and Invalid cards.
   Operating frequency is 13.56MHz
   RF protocol: ISO14443A
   Read and write distance: less than 10cm (regardless of reader)

2. NFC TAG Keychain – The blue Keychain tags set as Valid and Invalid tags.
   Main Chip: Philips Mifare 1 S50;
   Operating Frequency: 13.56 MHz;
   Read and write distance: 2.5 ~ 10cm

3. University Of Colorado Boulder – Buff One Card used to gain access to university secure areas.

- **4x3 Matrix Keypad**

A 4x3 matrix keypad has 7 pins, 3 columns and 4 rows. The keypad's buttons are connected in a matrix, so only 7 microcontroller pins are used to control the keypad. The keypad is used to enter the security and admin password in our system.

  - Security password:
    - One time override password to the system. It is set as 5678 in our system.

- Admin password:
    - Password to add a new card to the system for permanent access. It is set as 1234 in our system.
  - When a button is pressed, one of the rows is connected to one of the columns.
  - The CPU accesses both rows and columns through ports.
  - When a key is pressed, a row and a column make a contact.
  - To read a particular button in the matrix, the column pin is driven low using GPIO and then the corresponding row pin is read.
  - If the row pin reads low, then the button is pressed.
  - If the row pin reads high, then the button is not pressed.
  - '#' is used as a delimiter in our system to enter the security and admin password.

The hardware connections of the keypad with the STM32 board are mentioned in 2.4.1.

- **OLED**

SH1106 OLED chip consists of 132 segments, 64 commons that can support a maximum display resolution of 132 X 64. It is designed for Common Cathode type OLED panel. SH1106 embeds contrast control, display RAM oscillator and efficient DC-DC converter, which reduces the number of external components and power consumption.
The SH1106 works on I2C communication protocol for displaying data.
The hardware connections of the OLED with the STM32 board are mentioned in 2.4.6.
Operating voltage range: 1.65-3.5 volts

- **ISD 1820 voice recording and playback module**

The ISD1820 voice recording and playback module is a single-chip device that can record and play single message. It features non-volatile storage and can record for 8 to 20 seconds.
Steps to record and play audio:

  - Press the Record button to start recording.
  - An LED will blink every second to indicate that the recording is in progress.
  - Press the Record button again to stop recording.
  - Press the Play button to play the recorded audio.
  - The Play E button plays edge-triggered audio. The audio starts playing when the button is pressed and stops when the audio's edge is detected.
  - The Play L button plays level-triggered audio. The audio will play as long as the button is pressed and held, and stop when the button is released.

The hardware connections of the OLED with the STM32 board are mentioned in 2.4.3.
Supply Voltage: 3.3 volts

- Speaker

The Voice recording and playback module is connected to an 8 Ohm, 0.5 W speaker.
We have used the Adafruit Mini Metal Speaker.
The hardware connections of the Speaker with the ISD1820 module are mentioned in 2.4.4.

- **Buzzer**

Active buzzers produce sound directly when connected to a battery. So, they only require a DC power supply. The GPIO is pulled high when the buzzeris required to be ON and pulled low when it is expected to be OFF in our system.

The hardware connections of the OLED with the STM32 board are mentioned in 2.4.5.

2.3 Schematic

The schematic of the access system is shown in **Figure 4**. It is designed using the Ki-Cad EDA. Since we have used modules, the symbols for these components were not available in the tool. Hence, most of the symbols used in the schematic are designed by us in the Symbol editor.

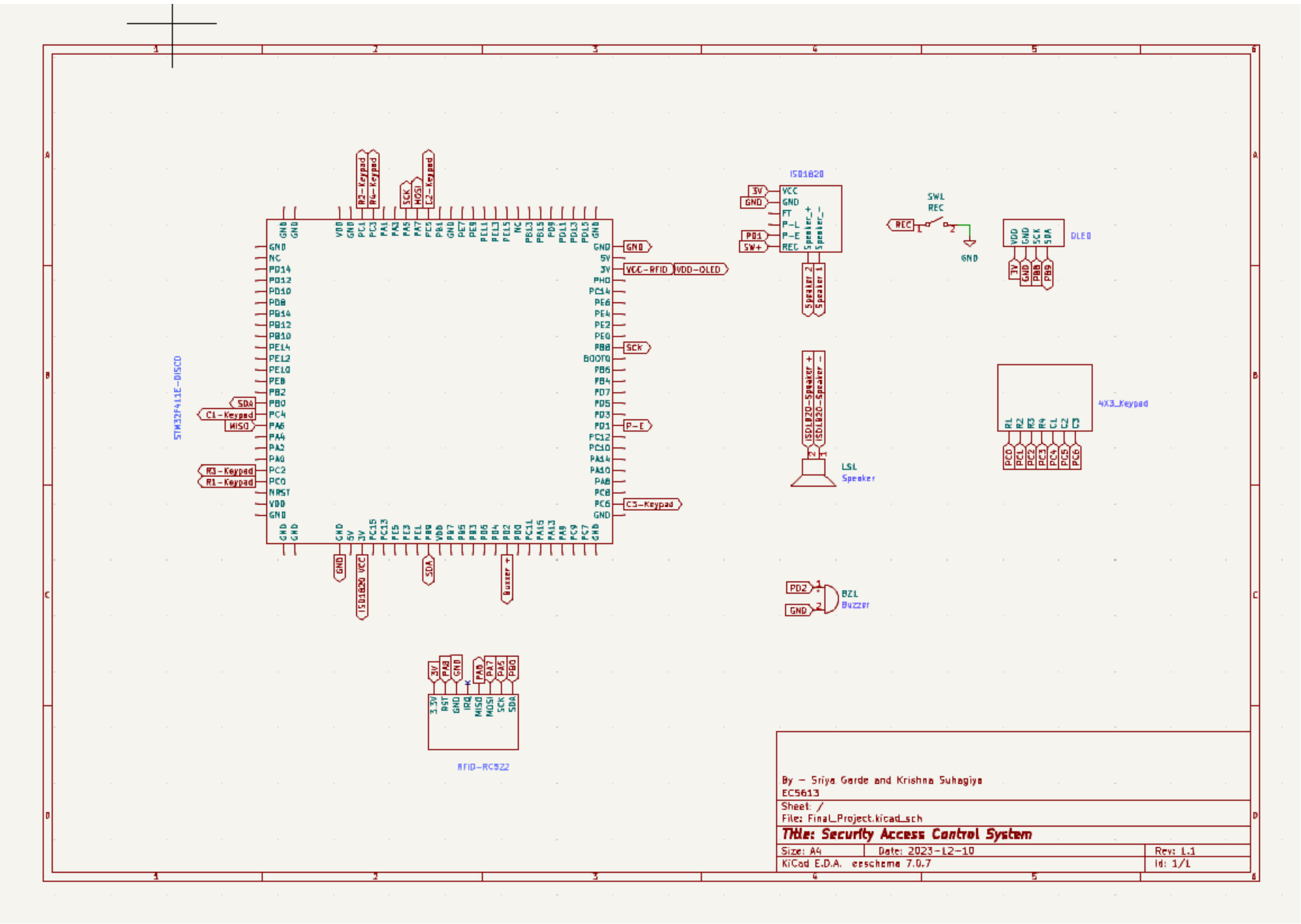


Figure 4: Schematic of Security Access Control System

2.4 Pin Connections

The tables below list the connections between the various system modules and STM32 microcontroller:

2.4.1 Keypad

Keypad	STM32
Row 1	PC0
Row 2	PC1
Row 3	PC2
Row 4	PC3

Column 1	PC4
Column 2	PC5
Column 3	PC6

2.4.2 MFRC 522

MFRC522	STM32
VCC	3V
RST	PA8
GND	PC2
MISO	PC3
MOSI	PC4
SCK	PC5
SDA	PC6
IRQ	NC

2.4.3 ISD1820 Voice Recording & Playback module

ISD1820	STM32
VCC	3V
GND	GND
P-E	PD1

2.4.4 Speaker

Speaker	ISD1820
+ Terminal	SPI +
- Terminal	SPI -

2.4.5 Buzzer

Buzzer	STM32
VCC	PD2
GND	GND

2.4.6 OLED

OLED	STM32
VCC	3V
GND	GND
SCK	PB8
SDA	PB9

2.5 Project Flowchart and Algorithm

The following flowchart portrays all the different ways our security system works with different inputs.

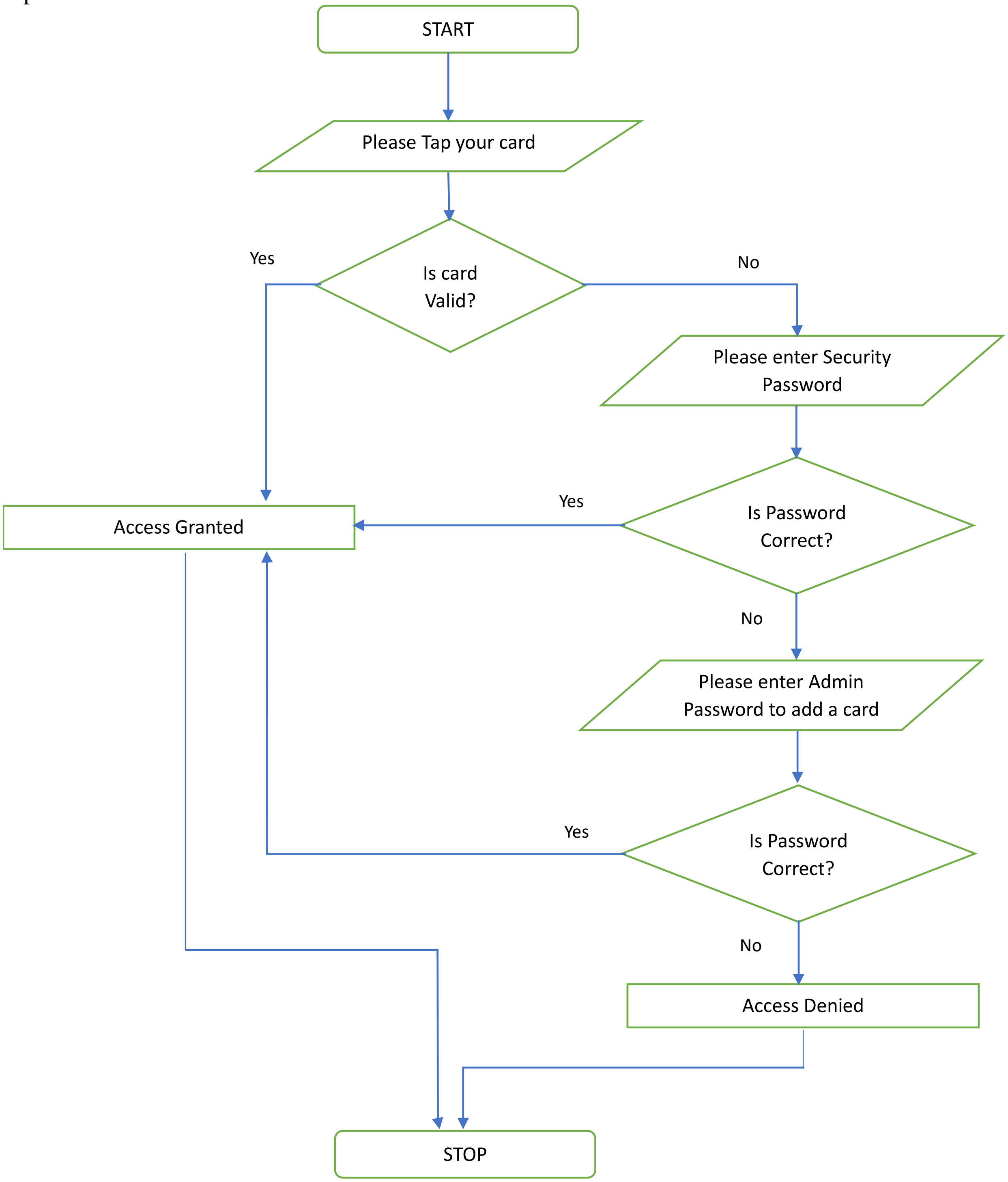
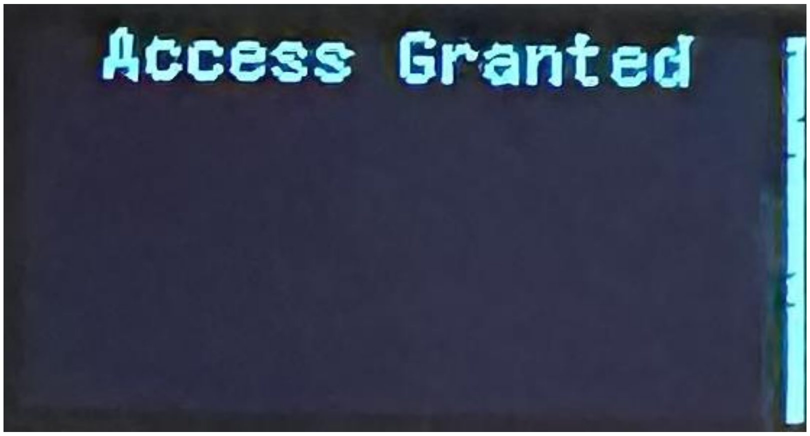


Figure 5: Flowchart of Security Access Control System

The flow can be distinctly divided into 4 cases:

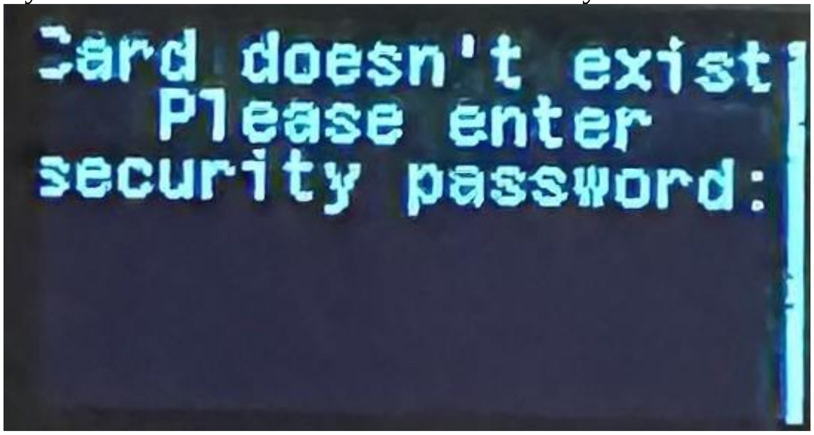
1. **Valid Card:**
   This is the happy scenario for the system. In this case, we tap a card which is registered as a valid card in the system..
   When the user taps this card, he gets immediate access because of his valid card.
   In this case, the Buzzer beeps whenever the user taps the card and OLED displays a message "Access Granted".



Access Granted

Figure 6: OLED screen with 'Access Granted' message

2. **Invalid Card (Security password):**
   In this case, the user's card is Invalid and hence he does not have access to the system. When the card is tapped, the Playback module plays an audio "Access Denied".
   The OLED displays "Card doesn't exist. Please Enter Security Password".

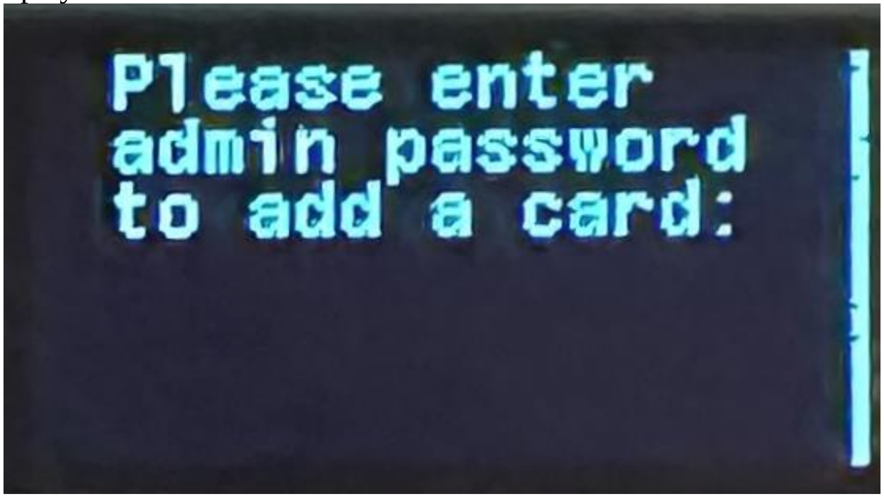


Card doesn't exist
Please enter
security password:

Figure 7: OLED screen displaying message asking for security password

The Security password is a One Time Override to the system. Entering the right password gives the user a one time access to the system. However, the next time the same card is used, it will not gain access because it is still an Invalid card.

3. **Invalid Card (Admin password):**
   This case also involves an Invalid card. In this case, the user is prompted for a Security password, but has entered an incorrect password.
   When the security password is incorrect, the user is asked for an Admin password.
   Then the Playback module plays an audio "Access Denied".
   The OLED displays "Please enter Admin Password to add a card".

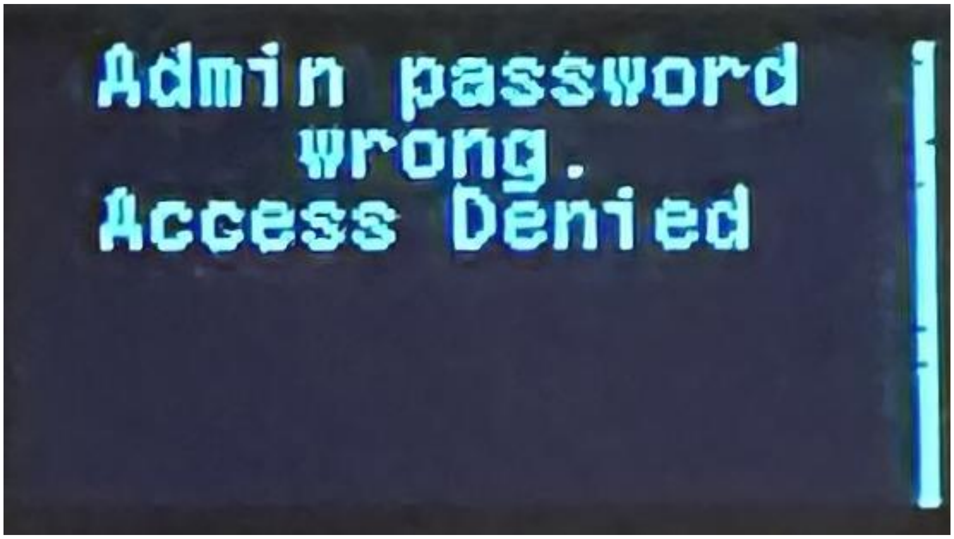


Please enter
admin password
to add a card:

Figure 8: OLED screen displaying a message asking for admin password

   The Admin password is used to add a new or invalid card to the system to make it a Valid card. When the correct Admin password is used, the card tapped is added to the system and becomes a Valid card. In this case, when the user taps the card henceforth, an access will be granted.

4. **Invalid Card (Access Denied)**
   When the user taps an Invalid card, he is asked to enter a Security password. If that password is wrong, an Admin password will be required to add the card as a Valid card.
   In case if this password is also entered wrong, the user is refused access to the system.
   The OLED displays – "Admin password wrong. Access Denied."



Admin password
wrong.
Access Denied

Figure 9: OLED screen indicating 'Access Denied' due to wrong admin password

The Playback module plays the “Access Denied” audio.
Now, the user has to start the process again by tapping the card on the reader.

2.6 Software Design

The following are the major two communication protocols used for developing the security access control system:

2.6.1 Serial Programming Interface (SPI)

The Serial Peripheral Interface (SPI) communication protocol serves as the conduit for data exchange between the STM32 microcontroller and the RFID-RC522 module. SPI is a synchronous serial communication protocol that facilitates the transmission of data between a master device (STM32) and slave (RFID-RC522).

The STM32 microcontroller acts as the master device in the SPI communication setup. It initiates and controls the data transfer process by generating clock signals and managing the data lines.
As the slave device, the RFID-RC522 module responds to commands and requests from the STM32 master.

SPI Communication Basics:

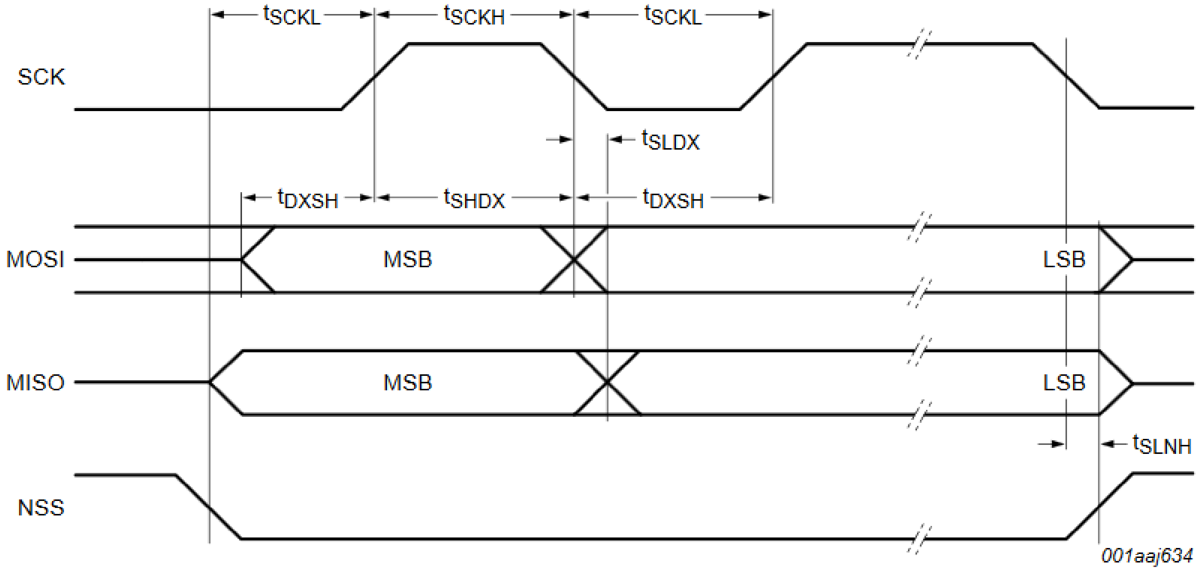
- **Clock (SCK):** The master (STM32) generates a clock signal (SCK) to synchronize data transfer.

Clock polarity and phase configurations may be adjusted based on the requirements of the connected devices.

- **Master Out Slave In (MOSI) and Master in Slave Out (MISO):**
    - **MOSI (STM32 to RFID-RC522):** The master sends data to the slave through this line.
    - **MISO (RFID-RC522 to STM32):** The slave responds by sending data back to the master.
- **Chip Select (CS/SS):** The Chip Select line designates the active slave device during communication. When low, it signals the RFID-RC522 module to listen for incoming data.

Communication Sequence:

- **Initiation:** The STM32 initializes communication by bringing the Chip Select line low. It configures the clock and data lines according to the required settings.
- **Data Transfer:** Data is transferred in frames, typically consisting of 8 bits. The master and slave exchange bits on the MOSI and MISO lines, respectively, with each clock cycle.
- **Termination:** The communication is concluded by bringing the Chip Select line high.



Remark: The signal NSS must be LOW to be able to send several bytes in one data stream. To send more than one data stream NSS must be set HIGH between the data streams.

Figure 10: SPI Communication timing diagram[4]

2.6.2 Inter-Integrated Circuit (I2C)

The Inter-Integrated Circuit (I2C) communication protocol serves as the communication bridge between the STM32 microcontroller and the SSD1106 OLED display in our project. I2C is a serial communication protocol that allows for the exchange of data between a master device (STM32) and OLED (SSD1106).

Key Components:

1. **STM32 Microcontroller:**
   - Acts as the master device in the I2C communication.
   - Initiates and controls the data exchange process with the OLED display.
2. **SSD1106 OLED Display:**
   - Operates as the slave device, responding to commands and data requests from the STM32 master.
   - Displays visual information on the OLED screen.

I2C Communication Basics:

1. **Bus Structure:**
   - I2C uses a two-wire bus system: SDA (Serial Data) and SCL (Serial Clock).
   - SDA carries the data, while SCL carries the clock signal for synchronization.
2. **Start and Stop Conditions:**

   - Communication begins with a Start condition and ends with a Stop condition.
   - The Start condition signifies the beginning of a data transfer session, and the Stop condition indicates the end.
3. **Addressing:**
   - Each I2C device has a unique 7-bit address.
   - The master sends the address of the slave it wishes to communicate with.
4. **Data Transfer:**
   - Data is transferred in 8-bit bytes, with each byte followed by an acknowledgment bit.
   - The master and slave devices take turns to send or receive data.

Communication Sequence:

1. **Initiation:**
   - The STM32 initiates communication by generating a Start condition on the I2C bus.
   - It specifies the address of the SSD1106 OLED display as the target.
2. **Addressing and Data Transfer:**
   - Following the Start condition, the master sends the address byte, indicating the SSD1106 as the intended recipient.
   - Subsequent data bytes are exchanged between the master and the slave.
3. **Termination:**
   - The communication session concludes with a Stop condition.

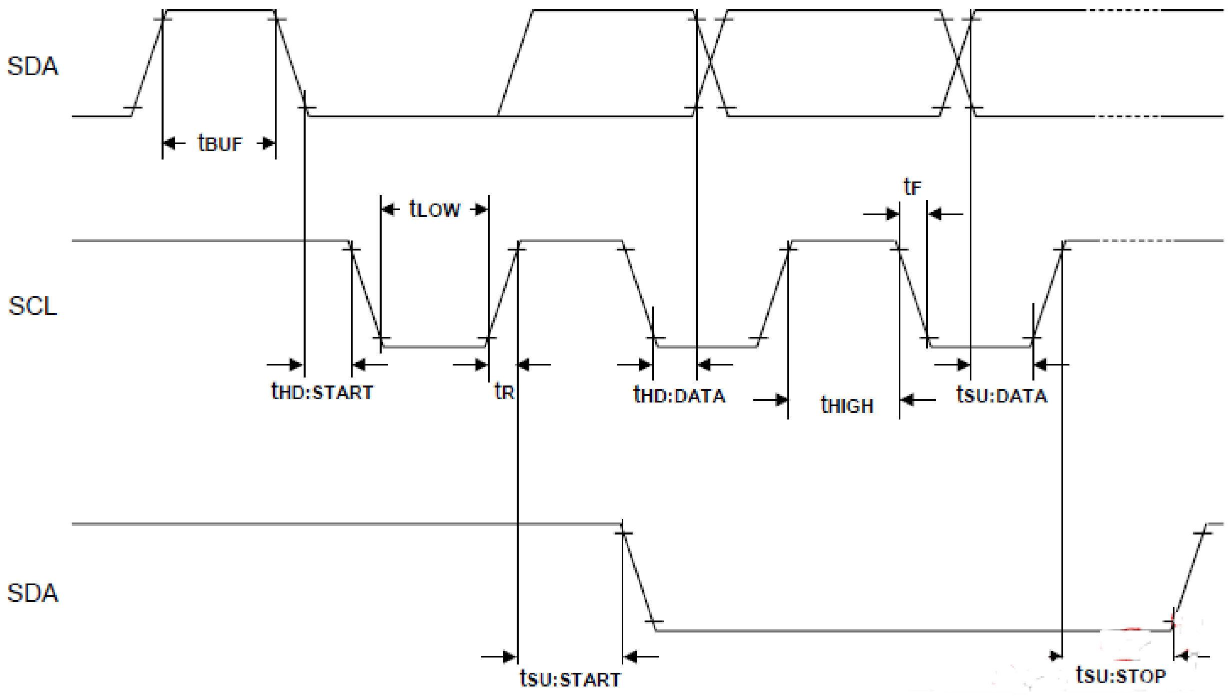


Figure 11: I2C communication timing diagram[5]

2.7 Testing Process

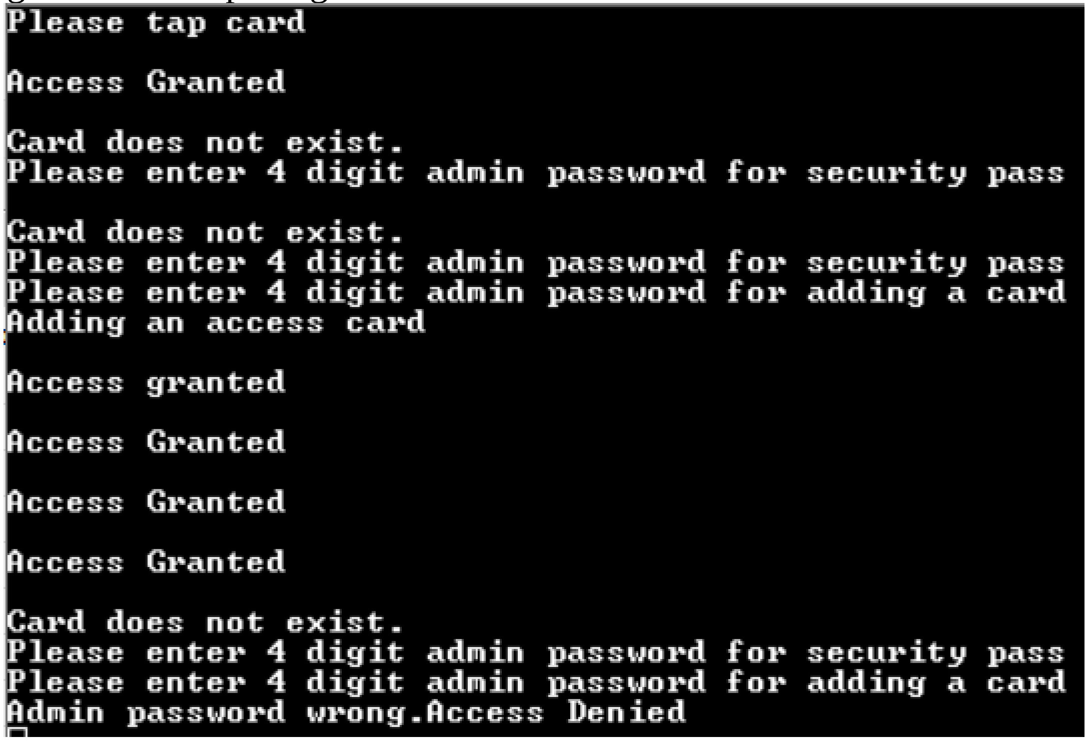
An incremental testing approach was followed for our security access control system.
To confirm that the hardware is working, we started with verifying RFID RC522, OLED and buzzer module with the Arduino and then moved to STM32 board.

We worked on separate elements and got them working with repetitive testing and solved any interfacing issues.

- It was started with the code development and testing of RFID RC522 module. First the UIDs of the RFID cards and keycards were fetched and then the code was developed to check the validity of the UIDs.
- Keyboard, OLED, Buzzer and voice recording and playback module were also tested separately.
- RFID RC522 module was validated then with the OLED.
- Furthermore, keypad inputs were validated with the buzzer and voice recording and playback modules.
- In the end, the whole system was integrated and validated.

Additionally, the serial terminal was setup by developing the APIs for transmitting data to UART to debug the system while development.

Following are the example logs on the serial terminal:



```
Please tap card

Access Granted

Card does not exist.
Please enter 4 digit admin password for security pass

Card does not exist.
Please enter 4 digit admin password for security pass
Please enter 4 digit admin password for adding a card
Adding an access card

Access granted

Access Granted

Access Granted

Access Granted

Card does not exist.
Please enter 4 digit admin password for security pass
Please enter 4 digit admin password for adding a card
Admin password wrong.Access Denied
```

Figure 12: Debug Logs on Serial Console

3 Results and Error Analysis

This section describes the challenges faced during the development and the methods used to fix them:

1. **Keypad:**
   While interfacing the keypad, we faced a lot of issues with Keys debouncing. We tried different methods to remove this issue. We added delays, worked with flags, and eventually solved this problem through trial and error of different delay values.

2. **Buzzer:**
   The buzzer we initially got was a Passive buzzer which requires PWM input to function. It does not give any output with just 5V and Ground connections. While interfacing this buzzer, we faced issues with setting a specific working frequency for PWM module in order to get the Buzzer to beep the required tone.
   Eventually, we decided that a passive buzzer was not required to generate the functionality we require of getting a beep for a valid Access card. Hence, we got an active buzzer which only requires a high voltage and ground connections. We connected the positive pin to a GPIO pin which went high when we wanted the buzzer to beep.

3. **Voice recording module:**
   We wanted to use multiple messages to play on the speaker based on the access status but getting such module was taking time. So, we used the ISD 1820 voice recording and playback module which can record a single message to record the denied access status message and represent the granted access status with the buzzer.

4 Conclusion

Through this project, we were able to work on our area of interest - an access control system. We were able to make the system fully working having an ability to enforce security with valid and invalid access cards, add valid cards to the system and give one-time access for specific users. We learned a lot about what goes into developing an actual Security system and all the Hardware and Software parameters one should consider.

We were able to explore Communication protocols like SPI and I2C to integrate various modules required to make the system functional. We also got a chance to delve into the specifications and limitations of the functional elements of a Secure system.

5 Future Development Ideas

- **Ability to create a database for multiple cards:**
  - An addition to this system can be to use Database platforms like SQL and connect it to the system to validate multiple cards for Access. The database can be accessed with Queries with the UID of each card as a key and access can be validated using that.
- **Ability to Delete cards from database:**
  - An additional password-controlled functionality of deleting cards from the system or setting an expiry date for the validity of the cards can be added.
- **Ability to store Valid cards data using some form of Key Encryption to protect unique user identities:**
  - An important part of Secure systems is protecting the unique user identities. A good addition to the system would be storing the UID-Keys of the access cards using some form of Encryption.
- **Ability to play multiple voice recorded messages:**
  - The current Voice Recording and playback module records and plays a single track. An expansion to make the system more interactive and inclusive would be to provide specific voice messages at multiple stages.

6 Contributions

Both team members worked on the RFID module interfacing with SPI driver.

Sriya worked on interfacing the Matrix Keypad, Buzzer and Voice Recording and playback module independently with the controller.

Krishna worked on interfacing the OLED module using I2C driver and the speaker module.

Sriya integrated the codes for Keypad, Buzzer and Voice recording module with STM32 and UART driver (used only in debug mode) and tested combined functionality.

Krishna integrated the codes for OLED Module and RFID Module.

Both worked on combining the two separate functional codes into one main system code for the entire functionality of the project.

Code Cleanup, Merging and Comments were added by Krishna.
The hardware schematic was designed by Sriya.
Both team members worked on the Hardware design and the project board as well as the Report.

7 Acknowledgements

We would like to thank Prof. Linden McClure for his support and for providing us with an opportunity to explore our technical areas of interest and work on them via this Project.

We would also like to extend our gratitude to the TAs for Fall 2023, for their help in framing the scope of the project and any technical support.

We would like to acknowledge and thank the staff of ITLL – CU Boulder for their help and suggestions in putting together the Hardware of the project and providing required materials.

We would like to appreciate our teamwork and express thanks to each other for our collaborative effort to make this project successful on time.

8 References

[1] https://www.st.com/resource/en/reference_manual/rm0383-stm32f411xce-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[2] https://www.st.com/resource/en/user_manual/um1842-discovery-kit-with-stm32f411ve-mcu-stmicroelectronics.pdf

[3] https://www.st.com/resource/en/data_brief/32f411ediscovery.pdf

[4] https://electronics.stackexchange.com/questions/325684/rfid-rc522-with-stm32f769i-discovery

[5] https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[6] https://cdn-shop.adafruit.com/datasheets/S50.pdf

[7] https://www.pololu.com/file/0J1813/SH1106.pdf

[8] https://github.com/thinkrobotics/DATASHEET/blob/master/ELECTRONIC_COMPONENTS/ELC7016/Datasheet%20for%20ELC7016.pdf

[9] https://components101.com/sites/default/files/component_datasheet/ISD1820-Module-Datasheet_0.pdf

[10] https://www.verical.com/datasheet/adafruit-speakers-1890-4292980.pdf

[11] https://www.electronicoscaldas.com/datasheet/LTE12-Series.pdf

[12] https://github.com/Hamid-R-Tanhaei/RFID-MIFARE-RC522-ARM-STM32/

[13] https://learn.parallax.com/tutorials/language/propeller-c/propeller-c-simple-devices/read-4x4-matrix-keypad

[14] https://www.youtube.com/watch?v=R5sv1hbONrk

[15] https://blog.embeddedexpert.io/?p=613

9 Appendices

Several appendices have been attached to this report in the order shown below.

9.1 Appendix - Bill of Materials

Part Name	**Manufacturer**	**Quantity**	**Cost**	**Source**
RFID RC522 RF SPI Card Sensor for Arduino module with 2 tags MFRC522 DC 3.3V USA	NXP	2	2.15	https://www.ebay.com/itm/224799456077
ISD1820 Voice Recording Playback Module Sound Recorder Board with Loudspeaker	Daier	1	3.45	https://www.ebay.com/itm/224262314287
1.3" I2C IIC 128X64 OLED Display Module Arduino Blue Color SSD1106 US	Adafruit	1	6.2	https://www.ebay.com/itm/324479771132
Keypad 4 x 3 Matrix Array 12 Key Arduino Membrane Switch Keyboard module	Adafruit	1	1.96	https://www.ebay.com/itm/324891098338
Active Buzzer	LTE12	1	NA	ITLL
PCB Board		1	7.79	ECEE-Electronics Store
STM32F411E-DISCO Board	STMicroelectronics	1		Received as a part of coursework
Shipping & Taxes			10.91	
TOTAL			**32.46**	

Note: If we were doing this project over again, we would have chosen APR33A3 instead of ISD 1820 voice recording and playback module because it supports multiple voice recording and playback.

9.2 Appendix - Schematics

The schematic of the system is available in ‘Schematics’ folder as well as at Figure 4.

9.3 Appendix - Firmware Source Code

The source code of the system is available in ‘Code’ folder as well as added below:

9.3.1 main.c

```
/******************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not
liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file   main.c
* @brief  A main file demonstrating the security access control system.
*
* @author Krishna Suhagiya and Sriya Garde
* @date   November 8, 2023
*
*/

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include "UART.h"
#include "delay.h"
#include "rfid.h"
#include "voice.h"
#include "beeper.h"
#include "oled.h"
#include "keypad.h"
#include "security_system_interface.h"

#define SIXTEEN_MHZ    16000000

int main(void) {
      systick_init_ms(SIXTEEN_MHZ);      // Initialize system clock
      beeper_init();                           // Initialize buzzer (beeper)
      RC522_init();                            // Initialize RFID reader module
      SSD1106_init();                          // Initialize OLED display
      SSD1106_gotoXY(0, 0);              // Set the cursor to (0,0) location on
the OLED
      init_keypad();                           // Initialize keypad
#ifdef DEBUG
      USART2_init();
      USART2_string_transmit("Please tap card \r\n");
#endif
      SSD1106_clear_screen();                 // Clear the OLED screen
      SSD1106_puts("Please tap card", &Font_7x10, 1);    // Set the default 
message to be displayed on OLED
      SSD1106_gotoXY(0, 10);            // Set the cursor to (0,10) location
      SSD1106_clear_line();             // Clear the line
      SSD1106_gotoXY(0, 20);            // Set the cursor to (0,20) location
      SSD1106_clear_line();             // Clear the line
      SSD1106_update_screen();          // Display the content set above on OLED

      while (1) {
            check_access();                   // Check the card access on every 
tap
      }
}
```

9.3.2 main.h

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not 
liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file   main.h
* @brief  A file declaring the error handler API.
*
* @author Krishna Suhagiya and Sriya Garde
* @date   November 8, 2023
* @revision 1.0
*
*/

#ifndef __MAIN_H
#define __MAIN_H

/**
 * @brief   A function for error handling.
 *
 * @param   None.
 *
 * @return  None.
 */
void Error_Handler(void);
```

```
#endif /* __MAIN_H */
```

9.3.3 UART.c

```
/*****************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file     UART.c
* @brief    A file defining the APIs for UART.
*
* @author   Sriya Garde
* @date     November 8, 2023
* @revision 1.0
*
*/

#ifdef DEBUG

#include "UART.h"

void USART2_init(void) {
      RCC->APB1ENR |= (1 << 17);
      RCC->AHB1ENR |= (1 << 0);

      GPIOA->MODER |= GPIO_MODER_MODER2_1 | GPIO_MODER_MODER3_1;

      GPIOA->AFR[0] |= (7 << 8) | (7 << 12);

      USART2->CR1 &= ~USART_CR1_UE;
      USART2->BRR = 0X0683;                        //9600 Baud rate 16 MHz

      USART2->CR1 = USART_CR1_TE | USART_CR1_RE | USART_CR1_UE;

}

char USART2_transmit(char input) {
      while (!(USART2->SR & USART_SR_TXE))
            ;                                            // Wait for transmit
      USART2->DR = input;
      return input;
}

char USART2_receive(void) {
      while (!(USART2->SR & USART_SR_RXNE))
            ;                                               // Wait for receive
      return (char) USART2->DR;

}

void USART2_string_transmit(char *text) {
      while (*text)
            USART2_transmit(*text++);          // Transmit character by character
}
#endif
```

9.3.4 UART.h

```
/*****************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file     UART.h
* @brief    A file declaring the APIs for UART.
*
* @author   Sriya Garde
* @date     November 8, 2023
* @revision 1.0
*
*/

#ifdef DEBUG
#ifndef UART_H_
#define UART_H_
#include "stm32f4xx.h"

#define RCC_GPIOA_ENR          (0b01)
#define RCC_GPIOD_ENR          (0b1 << 3)
#define RCC_TIM2_ENR           (0b01)
#define GPIOA_PORT0_INPUT      (0b11)
#define GPIOD_PORT12_OUTPUT    (0b01 << 24)
#define GPIOD_PORT15_OUTPUT    (0b01 << 30)
#define PORT12                 (0B01<<12)
#define PORT15                 (0B01<<15)

/**
 * @brief   A function to initialize UART2 for to receive or transmit characters
and strings.
 *
 * @param   None
 *
 * @return  None.
 */
void USART2_init(void);

/**
 * @brief   A function to transmit given character input.
 *
 * @param   Character to transmit
 *
 * @return  Transmitted character.
 */
char USART2_transmit(char input);

/**
 * @brief   A function to receive given character transmitted.
 *
 * @param   NULL
 *
 * @return  Character received.
 */
char USART2_receive(void);

/**
 * @brief   A function to receive given string transmitted.
 *
 * @param   Characters received through receive function
 *
 * @return  NULL
 */
void USART2_string_transmit(char *text);

#endif /* UART_H_ */
#endif
```

9.3.5 delay.c

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not
liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file   delay.c
* @brief  A file defining the delay related APIs.
*
```

```
* @author Krishna Suhagiya and Sriya Garde
* @date   November 8, 2023
* @revision 1.0
*
*/

#include "stm32f4xx.h"                  // Device header
#include "delay.h"

volatile uint32_t ms, rms;

void systick_init_ms(uint32_t freq) {
      __disable_irq();
      SysTick->LOAD = (freq / 1000) - 1;
      SysTick->VAL = 0;
      SysTick->CTRL = 7; //0b00000111;
      __enable_irq();
}

uint32_t millis(void) {
      __disable_irq();
      rms = ms; //store current ms in rms
      __enable_irq();
      return rms;
}

void SysTick_Handler(void) {
      ms++; // Increment the milliseconds counter
}

void delay(uint32_t ms) {
      uint32_t start = millis();

      do {
            ;
      } while (millis() - start < ms);

}
```

9.3.6 delay.h

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not
liable for
* any misuse of this material.
*
*****************************************************************************/
/**
```

```
* @file   delay.h
* @brief  A file declaring the delay related APIs.
*
* @author Krishna Suhagiya and Sriya Garde
* @date   November 8, 2023
* @revision 1.0
*/

#ifndef __DELAY_H
#define __DELAY_H

#include <stdint.h>

/**
 * @brief   A function to get the current time in milliseconds.
 *
 * @param   NULL
 *
 * @return  Current time.
 */
uint32_t millis(void);

/**
 * @brief   A function to initialize the systick.
 *
 * @param   The desired frequency is hertz.
 *
 * @return  None.
 */
void systick_init_ms(uint32_t freq);

/**
 * @brief   A function to delay the system by the specified milliseconds.
 *
 * @param   The desired delay in milliseconds.
 *
 * @return  None.
 */
void delay(uint32_t ms);

#endif /* __DELAY_H */
```

9.3.7 rfid.c

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification, or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde, and the University of Colorado are
not liable for
```

```
* any misuse of this material.
*
****************************************************************************/
/**
* @file     rfid.c
* @brief    A file defining the RFID (RC522) related APIs.
*
* @author   Krishna Suhagiya and Sriya Garde
* @date     November 8, 2023
* @reference https://github.com/Hamid-R-Tanhaei/RFID-MIFARE-RC522-ARM-STM32/
* @revision 1.0
*/

#include "stdio.h"
#include "RFID.h"
#include "SPI.h"
#include "stdbool.h"
#include "stm32f4xx.h"
#include "delay.h"

/*
 * STM32 ->RFID
 * SPI  -> SPI
 * PA8  ->RST
 * PB0  ->CS
 * */

// Function to initialize the RC522 RFID reader
void RC522_init(void) {
      // Initializing SPI communication between RFID reader and STM32
      spi_init();
      GPIOA->MODER |= GPIO_MODER_MODE8_0;
      GPIOA->MODER &= ~GPIO_MODER_MODE8_1;

      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;

      GPIOB->MODER |= GPIO_MODER_MODE0_0;

      GPIOA->BSRR = GPIO_BSRR_BR8;
      delay(50);
      GPIOA->BSRR = GPIO_BSRR_BS8;
      delay(50);
      RC522_reset();

      RC522_reg_write8(MFRC522_REG_T_MODE, 0x80); // Timer starts automatically
at the end of the transmission
      RC522_reg_write8(MFRC522_REG_T_PRESCALER, 0xA9); // The lower TPrescaler
value
      RC522_reg_write8(MFRC522_REG_T_RELOAD_L, 0xE8); // Lower 8 bits of the 16-
bit timer reload value
      RC522_reg_write8(MFRC522_REG_T_RELOAD_H, 0x03); // Higher 8 bits of the
16-bit timer reload value

      RC522_reg_write8(MFRC522_REG_TX_AUTO, 0x40);
      RC522_reg_write8(MFRC522_REG_MODE, 0x3D);

      RC522_antenna_ON();   // Open the antenna to read any RFID tags
}

// Function to control the state of the RFID CS pin
void RC522_spi_cs_write(bool state) {
      if (state) {
            GPIOB->ODR |= (1UL << 0);
      } else {
            GPIOB->ODR &= ~(1UL << 0);
      }
}

// Function to read a register (8 bits) from the RC522
uint8_t RC522_reg_read8(uint8_t reg) {
      RC522_spi_cs_write(0);
      reg = ((reg << 1) & 0x7E) | 0x80;
      spi_transmit(&reg, 1);
      uint8_t dataRd = 0;
      spi_receive(&dataRd, 1);
      RC522_spi_cs_write(1);
      return dataRd;
}

// Function to write a value (8 bits) to a register in the RC522
void RC522_reg_write8(uint8_t reg, uint8_t data8) {
      RC522_spi_cs_write(0);
      uint8_t txData[2] = { 0x7E & (reg << 1), data8 };
      spi_transmit(txData, 2);
      RC522_spi_cs_write(1);
}

// Function to set a specific bit in a register of the RC522
void RC522_set_bit(uint8_t reg, uint8_t mask) {
      RC522_reg_write8(reg, RC522_reg_read8(reg) | mask);
}

// Function to clear a specific bit in a register of the RC522
void RC522_clear_bit(uint8_t reg, uint8_t mask) {
      RC522_reg_write8(reg, RC522_reg_read8(reg) & (~mask));
}

// Function to reset the RC522
void RC522_reset(void) {
      RC522_reg_write8(0x01, 0x0F);
}

// Function to turn on the antenna for the RC522
void RC522_antenna_ON(void) {
      uint8_t temp;

      temp = RC522_reg_read8(MFRC522_REG_TX_CONTROL); // Output signal on pin
TX2
```

```
	if (!(temp & 0x03)) {
		RC522_set_bit(MFRC522_REG_TX_CONTROL, 0x03);
	}
}

// Function to check for an RFID card and retrieve its UID
bool RC522_check_card(uint8_t *id) {
	bool status = false;
	// Find cards if tapped against receiver
	status = RC522_request(PICC_REQIDL, id);
	if (status == true) {
		// If card is detected, Card detected
		// Return card UID 4 bytes
		status = RC522_anti_coll(id);
	}
	RC522_halt();		// Command card into hibernation

	return status;
}

// Function to request the RFID card and get its tag type
bool RC522_request(uint8_t reqMode, uint8_t *tagType) {
	bool status = false;
	uint16_t backBits;
	RC522_reg_write8(MFRC522_REG_BIT_FRAMING, 0x07);
	tagType[0] = reqMode;
	status = RC522_to_card(PCD_TRANSCEIVE, tagType, 1, tagType, &backBits);
	if ((status != true) || (backBits != 0x10)) {
		status = false;
	}
	return status;
}

// Function to transmit data to the RFID card and receive its response
bool RC522_to_card(uint8_t command, uint8_t *sendData, uint8_t sendLen,
		uint8_t *backData, uint16_t *backLen) {
	bool status = false;
	uint8_t irqEn = 0x00;
	uint8_t waitIRq = 0x00;
	uint8_t lastBits;
	uint8_t n;
	uint16_t i;

	irqEn = 0x77;
	waitIRq = 0x30;

	RC522_reg_write8(MFRC522_REG_COMM_IE_N, irqEn | 0x80);
	RC522_clear_bit(MFRC522_REG_COMM_IRQ, 0x80);
	RC522_set_bit(MFRC522_REG_FIFO_LEVEL, 0x80);

	RC522_reg_write8(MFRC522_REG_COMMAND, PCD_IDLE);

	// Writing data to the FIFO
	for (i = 0; i < sendLen; i++) {
```

```
            RC522_reg_write8(MFRC522_REG_FIFO_DATA, sendData[i]);
        }

        // Execute the command
        RC522_reg_write8(MFRC522_REG_COMMAND, command);
        if (command == PCD_TRANSCEIVE) {
            RC522_set_bit(MFRC522_REG_BIT_FRAMING, 0x80); // StartSend=1, 
transmission of data starts
        }

        // Waiting to receive data to complete
        i = 100; // i according to the clock frequency adjustment, the operator M1 
card maximum waiting time 25ms???
        do {
            // CommIrqReg[7..0]
            // Set1 TxIRq RxIRq IdleIRq HiAlerIRq LoAlertIRq ErrIRq TimerIRq
            n = RC522_reg_read8(MFRC522_REG_COMM_IRQ);
            i--;
        } while ((i != 0) && !(n & 0x01) && !(n & waitIRq));

        RC522_clear_bit(MFRC522_REG_BIT_FRAMING, 0x80);     // StartSend=0

        if (i != 0) {
            if (!(RC522_reg_read8(MFRC522_REG_ERROR) & 0x1B)) {
                status = true;
                if (n & irqEn & 0x01) {
                    status = false;
                }

                if (command == PCD_TRANSCEIVE) {
                    n = RC522_reg_read8(MFRC522_REG_FIFO_LEVEL);
                    uint8_t l = n;
                    lastBits = RC522_reg_read8(MFRC522_REG_CONTROL) & 0x07;
                    if (lastBits) {
                        *backLen = (n - 1) * 8 + lastBits;
                    } else {
                        *backLen = n * 8;
                    }

                    if (n == 0) {
                        n = 1;
                    }
                    if (n > MFRC522_MAX_LEN) {
                        n = MFRC522_MAX_LEN;
                    }

                    // Reading the received data in FIFO
                    for (i = 0; i < n; i++) {
                        uint8_t d = RC522_reg_read8(MFRC522_REG_FIFO_DATA);
                        if (l == 4)
                            printf("%02x ", d);
                        backData[i] = d;
                    }
                    if (l == 4)
```

```
                        printf("\r\n");
                    return status;
                }
            } else {
                printf("error\r\n");
                status = false;
            }
        }

        return status;
}

// Function to acquire the UID of the RFID card
bool RC522_anti_coll(uint8_t *serNum) {
        bool status;
        uint8_t i;
        uint8_t serNumCheck = 0;
        uint16_t unLen;

        RC522_reg_write8(MFRC522_REG_BIT_FRAMING, 0x00); // TxLastBists =
BitFramingReg[2..0]

        serNum[0] = PICC_ANTICOLL;
        serNum[1] = 0x20;
        status = RC522_to_card(PCD_TRANSCEIVE, serNum, 2, serNum, &unLen);

        if (status == true) {
            // Check card serial number
            for (i = 0; i < 4; i++) {
                serNumCheck ^= serNum[i];
            }
            if (serNumCheck != serNum[i]) {
                status = false;
            }
        }
        return status;
}

// Function to put the RFID card reader into hibernation until the card has been
processed
void RC522_halt(void) {
        uint16_t unLen;
        uint8_t buff[4];

        buff[0] = PICC_HALT;
        buff[1] = 0;
        RC522_calculate_CRC(buff, 2, &buff[2]);

        RC522_to_card(PCD_TRANSCEIVE, buff, 4, buff, &unLen);
}

// Function to calculate the CRC for RFID card communication
void RC522_calculate_CRC(uint8_t *pIndata, uint8_t len, uint8_t *pOutData) {
        uint8_t i, n;
```

```
      RC522_clear_bit(MFRC522_REG_DIV_IRQ, 0x04);     // CRCIrq = 0
      RC522_set_bit(MFRC522_REG_FIFO_LEVEL, 0x80);      // Clear the FIFO
pointer

      // Writing data to the FIFO
      for (i = 0; i < len; i++) {
            RC522_reg_write8(MFRC522_REG_FIFO_DATA, *(pIndata + i));
      }
      RC522_reg_write8(MFRC522_REG_COMMAND, PCD_CALCCRC);

      // Wait CRC calculation is complete
      i = 0xFF;
      do {
            n = RC522_reg_read8(MFRC522_REG_DIV_IRQ);
            i--;
      } while ((i != 0) && !(n & 0x04));      // CRCIrq = 1

      // Read CRC calculation result
      pOutData[0] = RC522_reg_read8(MFRC522_REG_CRC_RESULT_L);
      pOutData[1] = RC522_reg_read8(MFRC522_REG_CRC_RESULT_M);
}
```

9.3.8 rfid.h

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not
liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file   rfid.h
* @brief  A file declaring the RFID (RC522) related APIs.
*
* @author Krishna Suhagiya and Sriya Garde
* @date   November 8, 2023
* @reference https://github.com/Hamid-R-Tanhaei/RFID-MIFARE-RC522-ARM-STM32/
* @revision 1.0
*
*/

#ifndef __RFID_H
#define __RFID_H

#include "stdbool.h"
#include "stdint.h"
```

```
/* MFRC522 Commands */
#define PCD_IDLE            0x00   //NO action; Cancel the current command
#define PCD_AUTHENT           0x0E   //Authentication Key
#define PCD_TRANSCEIVE          0x0C   //Transmit and receive data,
#define PCD_CALCCRC           0x03   //CRC Calculate

/* Mifare_One card command word */
#define PICC_REQIDL           0x26   // find the antenna area does not enter 
hibernation
#define PICC_ANTICOLL         0x93   // anti-collision
#define PICC_HALT           0x50   // Sleep

/* MFRC522 Registers */
//Page 0: Command and Status
#define MFRC522_REG_COMMAND       0x01
#define MFRC522_REG_COMM_IE_N     0x02
#define MFRC522_REG_COMM_IRQ      0x04
#define MFRC522_REG_DIV_IRQ       0x05
#define MFRC522_REG_ERROR       0x06
#define MFRC522_REG_FIFO_DATA     0x09
#define MFRC522_REG_FIFO_LEVEL      0x0A
#define MFRC522_REG_CONTROL       0x0C
#define MFRC522_REG_BIT_FRAMING     0x0D
//Page 1: Command
#define MFRC522_REG_MODE        0x11
#define MFRC522_REG_TX_CONTROL      0x14
#define MFRC522_REG_TX_AUTO       0x15
#define MFRC522_REG_CRC_RESULT_M    0x21
#define MFRC522_REG_CRC_RESULT_L    0x22
#define MFRC522_REG_T_MODE        0x2A
#define MFRC522_REG_T_PRESCALER     0x2B
#define MFRC522_REG_T_RELOAD_H      0x2C
#define MFRC522_REG_T_RELOAD_L      0x2D

#define MFRC522_MAX_LEN         16

/**
 * @brief   A function to initialize RC522 RFID module.
 *
 * @param   None
 *
 * @return  None.
 */
void RC522_init(void);

/**
 * @brief   A function to control the Chip Select (CS) pin of the RC522 module.
 *
 * @param   state to set pin low/high
 *
 * @return  None.
 */
void RC522_spi_cs_write(bool state);
```

```
/**
 * @brief   A function to read an 8-bit register from the RC522 RFID/NFC module
using SPI communication.
 *
 * @param   Register address
 *
 * @return  Received data.
 */
uint8_t RC522_reg_read8(uint8_t reg);

/**
 * @brief   A function to write an 8-bit value to a specific register in the
RC522 RFID/NFC module using SPI communication.
 *
 * @param   reg Register address
 *          data8 An 8-bit value to write
 *
 * @return  Data value.
 */
void RC522_reg_write8(uint8_t reg, uint8_t data8);

/**
 * @brief   A function to set specific bits in a register of the RC522 RFID/NFC
module.
 *
 * @param   reg Register address
 *          mask Mask value
 *
 * @return  Data value.
 */
void RC522_set_bit(uint8_t reg, uint8_t mask);

/**
 * @brief   A function to clear specific bits in a register of the RC522
RFID/NFC module.
 *
 * @param   reg Register address
 *          mask Mask value
 *
 * @return  None.
 */
void RC522_clear_bit(uint8_t reg, uint8_t mask);

/**
 * @brief   A function to reset the RC522 RFID/NFC module.
 *
 * @param   None
 *
 * @return  None.
 */
void RC522_reset(void);

/**
```

```
 * @brief   A function to turn on the antenna of the RC522 RFID/NFC module by
configuring the TX_CONTROL register.
 *
 * @param   None
 *
 * @return  None.
 */
void RC522_antenna_ON(void);

/**
 * @brief   A function to check for the presence of a card and retrieves its
Unique IDentifier (UID) if a card is detected.
 *
 * @param   id Pointer the card UID.
 *
 * @return  Card checking result.
 */
bool RC522_check_card(uint8_t *id);

/**
 * @brief   A function part of the process of making a request to a nearby
RFID/NFC card using the RC522 module.
 * It initiates the communication and requests the card to respond, then checks
the response to determine if a card has been detected.
 *
 * @param   reqMode Request mode.
 *          tagType Pointer to the tag type array.
 *
 * @return  Request result.
 */
bool RC522_request(uint8_t reqMode, uint8_t *tagType);

/**
 * @brief   A function to send a command to the card and receive the response
data.
 *
 * @param   command  command
 *          sendData Pointer to the data to send.
 *          sendLen  Length of the data to send.
 *          backData Pointer to the response data.
 *          backLen  Lenth of the response data.
 *
 * @return  Result.
 */
bool RC522_to_card(uint8_t command, uint8_t *sendData, uint8_t sendLen,
            uint8_t *backData, uint16_t *backLen);

/**
 * @brief   A function of anti-collision anti-collision to detect and select a
specific card when multiple cards are present in the reader's field.
 *
 * @param   serNum Pointer to the serial number array for anti-collision
 *
 * @return  Status of the anti-collision process.
 */
bool RC522_anti_coll(uint8_t *serNum);

/**
 * @brief   A function to halt communication with an RFID/NFC card using the 
RC522 module.
 *
 * @param   None
 *
 * @return  None.
 */
void RC522_halt(void);

/**
 * @brief   A function to calculate the CRC (Cyclic Redundancy Check) for a 
given set of input data using the RC522 RFID/NFC module.
 *
 * @param   pIndata Pointer to the input data
 *          len     Length of the input data
 *          pOutData Pointer to the output data
 *
 * @return  None.
 */
void RC522_calculate_CRC(uint8_t *pIndata, uint8_t len, uint8_t *pOutData);

#endif /* __RFID_H */
```

9.3.9 spi.c

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification, or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde, and the University of Colorado are 
not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file    spi.c
* @brief   A file defining the SPI communication protocol related APIs for 
communication between RC522 and STM32.
*
* @author  Krishna Suhagiya and Sriya Garde
* @date    November 8, 2023
* @revision 1.0
*/

#include <stdio.h>
#include <stdint.h>
#include "spi.h"
#include "stm32f4xx.h"
```

```
#include "delay.h"

#define AF5 0x05

// Function to initialize the SPI communication
void spi_init(void) {
      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // Enable clock for GPIOA
      RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // Enable clock for SPI1

      // Set alternate function mode for GPIO pins 5, 6, 7
      GPIOA->MODER |= GPIO_MODER_MODE5_1 | GPIO_MODER_MODE6_1 |
GPIO_MODER_MODE7_1;
      GPIOA->MODER &= ~(GPIO_MODER_MODE5_0 | GPIO_MODER_MODE6_0 |
GPIO_MODER_MODE7_0);

      // Set pin 5, 6, 7 (SCK, MISO, MOSI) to high-speed mode
      GPIOA->OSPEEDR |= GPIO_OSPEEDER_OSPEEDR5 | GPIO_OSPEEDER_OSPEEDR6 |
GPIO_OSPEEDER_OSPEEDR7;

      // Sets the alternate function to the pins to be used for SPI
Communication
      GPIOA->AFR[0] |= (AF5 << 20) | (AF5 << 24) | (AF5 << 28);

      // Clears the control register 2 of SPI1
      SPI1->CR2 = 0;

      // Set SPI1 as Master, Set Baud Rate, Enable s/w Slave Management, Enable
SPI1
      SPI1->CR1 = SPI_CR1_SSM | SPI_CR1_MSTR | SPI_CR1_BR_2 | SPI_CR1_SSI |
SPI_CR1_SPE;
}

// Function to transmit data over SPI
int8_t spi_transmit(uint8_t *data, uint32_t size) {
      uint8_t i = 0;
      uint32_t start = millis();

      // Used to check the value in the data register before transmission &
clear it
      if (SPI1->DR) {}

      // Used to check the status register & ensure it is ready for SPI transmit
operation
      if (SPI1->SR) {}

      while (i < size) {
            while (!((SPI1->SR) & SPI_SR_TXE)) {
                  if (millis() - start > 1000) { // Wait for transmit buffer to
be empty
                        printf("TXE timed out\r\n");
                        return -1;
                  }
            }
            SPI1->DR = data[i];  // Transmit data byte by byte

            while (!(SPI1->SR & SPI_SR_BSY)) {
                  if (millis() - start > 1000) {
                        printf("BSY timed out\r\n");
                        return -1;
                  }
            }

            i++;
      }

      // Wait for Transmit buffer to be empty
      while (!((SPI1->SR) & SPI_SR_TXE)) {
            if (millis() - start > 1000) {
                  printf("TXE2 timed out\r\n");
                  return -1;
            }
      }

      // Wait for transmit
      while ((SPI1->SR) & SPI_SR_BSY) {
            if (millis() - start > 1000) {
                  printf("BSY2 timed out\r\n");
                  return -1;
            }
      }

      // Used to check the value in data register after transmission & clear it
      if (SPI1->DR) {}

      // Used to check the status register & ensure it is done with SPI receive
operation
      if (SPI1->SR) {}

      return 0;
}

// Function to receive data over SPI
int8_t spi_receive(uint8_t *data, uint32_t size) {
      while (size) {
            uint32_t start = millis();
            SPI1->DR = 0;

            while (!(SPI1->SR & SPI_SR_RXNE)) {
                  if (millis() - start > 200) {
                        return -1;
                  }
            }

            *data++ = (SPI1->DR);
            size--;
      }
      return 0;
}
```

9.3.10 spi.h

```
/******************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not
liable for
* any misuse of this material.
*
******************************************************************************/
/**
* @file   spi.h
* @brief  A file declaring the SPI communication protocol related APIs for
communication between RC522 and STM32.
*
* @author Krishna Suhagiya and Sriya Garde
* @date   November 8, 2023
* @revision 1.0
*
*/

#ifndef __SPI_H
#define __SPI_H

#include <stdint.h>

/**
 * @brief   A function to initialize the SPI.
 *
 * @param   None
 *
 * @return  None.
 */
void spi_init(void);

/**
 * @brief   A function to transmit data over the SPI (Serial Peripheral
Interface) bus.
 *
 * @param   data Pointer to the data to transmit
 *          size Size of the data to transmit
 *
 * @return  Transmission status.
 */
int8_t spi_transmit(uint8_t *data, uint32_t size);

/**
```

```
 * @brief   A function to receive data over the SPI (Serial Peripheral
Interface) bus.
 *
 * @param   data Pointer to the data to receive
 *          size Size of the data to receive
 *
 * @return  Reception status.
 */
int8_t spi_receive(uint8_t *data, uint32_t size);

#endif /* __SPI_H */
```

9.3.11 keypad.c

```
/*****************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file      keypad.c
* @brief     A file defining the APIs for 4x3 matrix keypad.
*
* @author    Sriya Garde
* @date      November 18, 2023
* @reference https://www.youtube.com/watch?v=R5sv1hbONrk
*            https://learn.parallax.com/tutorials/language/propeller-
c/propeller-c-simple-devices/read-4x4-matrix-keypad
* @revision  1.0
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include "UART.h"
#include "keypad.h"
#include "delay.h"

volatile uint8_t buttonState[3][3] = { 0 };
uint32_t volatile *RCC_Base_Addr = (uint32_t*) 0x40023830;
uint32_t volatile *GPIOModer = (uint32_t*) 0x40020800;

uint32_t volatile *GPIOOutput = (uint32_t*) 0x40020814;
uint32_t volatile *GPIOIntput = (uint32_t*) 0x40020810;
uint32_t volatile *GPIOPullup = (uint32_t*) 0x4002080C;
```

```
uint32_t volatile *GPIOOspeed = (uint32_t*) 0x40020808;

char key_data[50] = { 0 };

void init_keypad(void) {

      //Base address RCC 0x4002 3800 + Address offset of RCC AHB1 = 0x30
      //Base address GPIOE 0x4002 1000 + Address offset of GPIO Moder = 0x00
      //GPIO Port o/p data register = 0x14

      *RCC_Base_Addr |= 1 << 2; //Setting the 4th bit of AHB1ENR for giving
clock to port C

      *GPIOModer &= ~(0xFF << 8); //Cols as Input , so 00 to E4,E5,E6 as input

      //*GPIOModer &= ~(0xFF<<0); //Clearing the bits before setting them
      *GPIOModer |= (0x55 << 0); //01010101 to set Output for E0,E1,E2,E3 pins

      *GPIOOspeed &= ~(0xFF << 0);

      *GPIOPullup &= ~(0xFF << 8);
      *GPIOPullup |= (0x15 << 8);

}

char* check_key(void) {
//****************************************ROW
1****************************************************
      char ch = '\0';
      bool button_pressed = false;
      int i = 0;
      memset(key_data, 0, 50);

      while (1) {
            *GPIOOutput |= (0xFF << 0);  // Set the value

            *GPIOOutput &= ~(1 << 0);    // 0th row

            if (!(*GPIOIntput & (1 << 4)) && !buttonState[0][0]) {   // Check
GPIO state
                  button_pressed = true;
                  ch = '1';
                  buttonState[0][0] = 1;
                  delay(50); // Debounce delay

            } else if (*GPIOIntput & (1 << 4)) {
                  buttonState[0][0] = 0;
            }

            if (!(*GPIOIntput & (1 << 5)) && !buttonState[0][1]) {   // Check
GPIO state
                  button_pressed = true;
                  ch = '2';
                  buttonState[0][1] = 1;
```

```
                delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 5)) {
                buttonState[0][1] = 0;
            }

            if (!(*GPIOIntput & (1 << 6)) && !buttonState[0][2]) {    // Check
GPIO state
                button_pressed = true;
                ch = '3';
                buttonState[0][2] = 1;
                delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 6)) {
                buttonState[0][2] = 0;
            }

//*******************************************ROW
2*********************************************************

            *GPIOOutput |= (0x0F << 0);

            *GPIOOutput &= ~(1 << 1);    // 1st row

            if (!(*GPIOIntput & (1 << 4)) && !buttonState[1][0]) {    // Check
GPIO state
                button_pressed = true;
                ch = '4';
                buttonState[1][0] = 1;
                delay(50); // Debounce delay

            } else if (*GPIOIntput & (1 << 4)) {
                buttonState[1][0] = 0;
            }

            if (!(*GPIOIntput & (1 << 5)) && !buttonState[1][1]) {    // Check
GPIO state
                button_pressed = true;
                ch = '5';
                buttonState[1][1] = 1;
                delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 5)) {
                buttonState[1][1] = 0;
            }

            if (!(*GPIOIntput & (1 << 6)) && !buttonState[1][2]) {    // Check
GPIO state
                button_pressed = true;
                ch = '6';
                buttonState[1][2] = 1;
                delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 6)) {
                buttonState[1][2] = 0;
            }
```

```
//*****************************************ROW
3*********************************************************
            *GPIOOutput |= (0x0F << 0);

            *GPIOOutput &= ~(1 << 2);    // 2nd row

            if (!(*GPIOIntput & (1 << 4)) && !buttonState[2][0]) {    // Check
GPIO state
                  button_pressed = true;
                  ch = '7';
                  buttonState[2][0] = 1;
                  delay(50); // Debounce delay

            } else if (*GPIOIntput & (1 << 4)) {
                  buttonState[2][0] = 0;
            }

            if (!(*GPIOIntput & (1 << 5)) && !buttonState[2][1]) {    // Check
GPIO state
                  button_pressed = true;
                  ch = '8';
                  buttonState[2][1] = 1;
                  delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 5)) {
                  buttonState[2][1] = 0;
            }

            if (!(*GPIOIntput & (1 << 6)) && !buttonState[2][2]) {    // Check
GPIO state
                  button_pressed = true;
                  ch = '9';
                  buttonState[2][2] = 1;
                  delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 6)) {
                  buttonState[2][2] = 0;
            }

//*****************************************ROW
4*********************************************************

            *GPIOOutput |= (0x0F << 0);

            *GPIOOutput &= ~(1 << 3);    // 4th row

            if (!(*GPIOIntput & (1 << 4)) && !buttonState[3][0]) {    // Check
GPIO state
                  //USART2_StringTransmit("* \r\n");
                  button_pressed = true;
                  ch = '*';
                  buttonState[3][0] = 1;
                  delay(50); // Debounce delay

            } else if (*GPIOIntput & (1 << 4)) {
                  buttonState[3][0] = 0;
            }

            if (!(*GPIOIntput & (1 << 5)) && !buttonState[3][1]) {   // Check
GPIO state
                  //USART2_StringTransmit("0 \r\n");
                  button_pressed = true;
                  ch = '0';
                  buttonState[3][1] = 1;
                  delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 5)) {
                  buttonState[3][1] = 0;
            }

            if (!(*GPIOIntput & (1 << 6)) && !buttonState[3][2]) {   // Check
GPIO state
                  //USART2_StringTransmit("# \r\n");
                  button_pressed = true;
                  ch = '#';
                  buttonState[3][2] = 1;
                  delay(50); // Debounce delay
            } else if (*GPIOIntput & (1 << 6)) {
                  buttonState[3][2] = 0;
            }
            if (ch == '#')
                  break;
            // Break if delimiter detected
            if ((button_pressed == true) && (!((!(ch == '*')) || (ch == '#'))))
{
#ifdef DEBUG
                  USART2_string_transmit(
                              "Invalid input. Please enter digits only.\r\n");
#endif
            } else if ((button_pressed == true) && ((ch >= '0') && (ch <= '9')))
{
                  key_data[i] = ch;
                  i++;
                  ch = '\0';
            }
      }
      return key_data;

}
```

9.3.12 keypad.h

```
/*****************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
******************************************************************************/
/**
* @file      keypad.h
* @brief     A file declaring the APIs for 4x3 matrix keypad.
*
* @author    Sriya Garde
* @date      November 18, 2023
* @revision  1.0
* @reference https://www.youtube.com/watch?v=R5sv1hbONrk
*            https://learn.parallax.com/tutorials/language/propeller-
c/propeller-c-simple-devices/read-4x4-matrix-keypad
*
*/

#ifndef __KEYPAD_H
#define __KEYPAD_H
#include "stm32f4xx.h"

/**
 * @brief   A function to initialize the keypad.
 *
 * @param   None.
 *
 * @return  None.
 */
void init_keypad(void);

/**
 * @brief   A function to check the correctness of the entered keys for
password.
 *
 * @param   None.
 *
 * @return  Pointer to the key inputs.
 */
char* check_key(void);

#endif /* __KEYPAD_H */
```

9.3.13 oled.c

```
/******************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya and the University of Colorado are not liable for
* any misuse of this material.
*
```

```
*****************************************************************************/
/**
* @file      oled.c
* @brief     A file defining the OLED SSD 1106 display related APIs.
*
* @author    Krishna Suhagiya
* @date      November 21, 2023
* @reference https://blog.embeddedexpert.io/?p=613
* @revision 1.0
*
*/

#include "oled.h"

//#define SSD1106_I2C_ADDR 0x3C
/* Write command */
#define SSD1106_WRITECOMMAND(command)      i2c_write_byte(SSD1106_I2C_ADDR,
0x00, (command))

/* SSD1106 data buffer */
static char SSD1106_Buffer[SSD1106_WIDTH * SSD1106_HEIGHT / 8];

/* Private SSD1106 structure */
typedef struct {
      uint16_t CurrentX;
      uint16_t CurrentY;
      uint8_t Initialized;
} SSD1106_t;

/* Private variable */
static SSD1106_t SSD1106;

#define SSD1106_DEACTIVATE_SCROLL                    0x2E // Stop scroll

uint8_t SSD1106_init(void) {

      /* Init I2C */
      i2c_init();
      SSD1106_i2c_init();

      /* A little delay */
      uint32_t p = 2500;
      while (p > 0)
            p--;

      /* Init LCD */
      SSD1106_WRITECOMMAND(0xAE); //display off
      SSD1106_WRITECOMMAND(0x20); //Set Memory Addressing Mode
      SSD1106_WRITECOMMAND(0x10); //00,Horizontal Addressing Mode;01,Vertical
Addressing Mode;10,Page Addressing Mode (RESET);11,Invalid
      SSD1106_WRITECOMMAND(0xB0); //Set Page Start Address for Page Addressing
Mode,0-7
      SSD1106_WRITECOMMAND(0xC8); //Set COM Output Scan Direction
      SSD1106_WRITECOMMAND(0x00); //set low column address
      SSD1106_WRITECOMMAND(0x10); //set high column address
      SSD1106_WRITECOMMAND(0x40); //set start line address
      SSD1106_WRITECOMMAND(0x81); //set contrast control register
      SSD1106_WRITECOMMAND(0xFF);
      SSD1106_WRITECOMMAND(0xA1); //set segment re-map 0 to 127
      SSD1106_WRITECOMMAND(0xA6); //set normal display
      SSD1106_WRITECOMMAND(0xA8); //set multiplex ratio(1 to 64)
      SSD1106_WRITECOMMAND(0x3F); //
      SSD1106_WRITECOMMAND(0xA4); //0xa4,Output follows RAM content;0xa5,Output
ignores RAM content
      SSD1106_WRITECOMMAND(0xD3); //set display offset
      SSD1106_WRITECOMMAND(0x00); //not offset
      SSD1106_WRITECOMMAND(0xD5); //set display clock divide ratio/oscillator
frequency
      SSD1106_WRITECOMMAND(0xF0); //set divide ratio
      SSD1106_WRITECOMMAND(0xD9); //set pre-charge period
      SSD1106_WRITECOMMAND(0x22); //
      SSD1106_WRITECOMMAND(0xDA); //set com pins hardware configuration
      SSD1106_WRITECOMMAND(0x12);
      SSD1106_WRITECOMMAND(0xDB); //set vcomh
      SSD1106_WRITECOMMAND(0x20); //0x20,0.77xVcc
      SSD1106_WRITECOMMAND(0x8D); //set DC-DC enable
      SSD1106_WRITECOMMAND(0x14); //
      SSD1106_WRITECOMMAND(0xAF); //turn on SSD1106 panel

      SSD1106_WRITECOMMAND(SSD1106_DEACTIVATE_SCROLL);

      SSD1106_fill(SSD1106_COLOR_BLACK);// Clear screen

      SSD1106_update_screen();     // Update screen

      /* Set default values */
      SSD1106.CurrentX = 0;
      SSD1106.CurrentY = 0;

      SSD1106.Initialized = 1;     // Initialized OK

      return SSD1106.Initialized;
}

void SSD1106_update_screen(void) {
      uint8_t m;

      for (m = 0; m < 8; m++) {
            SSD1106_WRITECOMMAND(0xB0 + m);
            SSD1106_WRITECOMMAND(0x00);
            SSD1106_WRITECOMMAND(0x10);

            /* Write multi data */
            SSD1106_i2c_write_multi(SSD1106_I2C_ADDR, 0x40,
                       &SSD1106_Buffer[SSD1106_WIDTH * m], SSD1106_WIDTH);
      }
}
```

```
void SSD1106_fill(SSD1106_COLOR_t color) {
      /* Set memory */
      memset(SSD1106_Buffer, (color == SSD1106_COLOR_BLACK) ? 0x00 : 0xFF,
                  sizeof(SSD1106_Buffer));
}

void SSD1106_draw_pixel(uint16_t x, uint16_t y, SSD1106_COLOR_t color) {
      if (x >= SSD1106_WIDTH || y >= SSD1106_HEIGHT) {
            return;
      }

      /* Set color */
      if (color == SSD1106_COLOR_WHITE) {
            SSD1106_Buffer[x + (y / 8) * SSD1106_WIDTH] |= 1 << (y % 8);
      } else {
            SSD1106_Buffer[x + (y / 8) * SSD1106_WIDTH] &= ~(1 << (y % 8));
      }
}

void SSD1106_gotoXY(uint16_t x, uint16_t y) {
      /* Set write pointers */
      SSD1106.CurrentX = x;
      SSD1106.CurrentY = y;
}

char SSD1106_putc(char ch, FontDef_t *Font, SSD1106_COLOR_t color) {
      uint32_t i, b, j;

      /* Check available space in LCD */
      if (
      SSD1106_WIDTH <= (SSD1106.CurrentX + Font->FontWidth) ||
      SSD1106_HEIGHT <= (SSD1106.CurrentY + Font->FontHeight)) {
            return 0;
      }

      /* Go through font */
      for (i = 0; i < Font->FontHeight; i++) {
            b = Font->data[(ch - 32) * Font->FontHeight + i];
            for (j = 0; j < Font->FontWidth; j++) {
                  if ((b << j) & 0x8000) {
                        SSD1106_draw_pixel(SSD1106.CurrentX + j,
(SSD1106.CurrentY + i),
                                    (SSD1106_COLOR_t) color);
                  } else {
                        SSD1106_draw_pixel(SSD1106.CurrentX + j,
(SSD1106.CurrentY + i),
                                    (SSD1106_COLOR_t) !color);
                  }
            }
      }

      /* Increase pointer */
      SSD1106.CurrentX += Font->FontWidth;
      /* Return character written */
      return ch;
}

char SSD1106_puts(char *str, FontDef_t *Font, SSD1106_COLOR_t color) {
      /* Write characters */
      while (*str) {
            /* Write character by character */
            if (SSD1106_putc(*str, Font, color) != *str) {
                  /* Return error */
                  return *str;
            }

            str++;
      }

      return *str;
}

void SSD1106_clear_screen(void) {
      SSD1106_fill(0);                  // Fill the buffer with zeros
      SSD1106_update_screen();          // Update the screen with the filled 
buffer
}

void SSD1106_clear_line(void) {
      SSD1106_puts("                  ", &Font_7x10, 1);  // Clear the whole 
line from the current cursor position
}

void SSD1106_i2c_init() {

      uint32_t p = 250000;              // Wait for I2C initialization
      while (p > 0)
            p--;

}

void SSD1106_i2c_write_multi(uint8_t address, uint8_t reg, char *data, 
            uint16_t count) {
      i2c_write_multi(address, reg, data, count);        // I2C write multi-
registers
}
```

9.3.14 oled.h

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file   oled.h
* @brief  A file declaring the OLED SSD 1106 display related APIs.
*
* @author Krishna Suhagiya
* @date   November 21, 2023
* @reference https://blog.embeddedexpert.io/?p=613
* @revision 1.0
*
*/

#ifndef __OLED_H
#define __OLED_H

#include "stm32f4xx.h"
#include <stdlib.h>
#include <string.h>
#include "fonts.h"
#include "i2c.h"

/**
 * This SSD1106 LCD uses I2C for communication
 *
 * Default pinout
 *
 SSD1106    |STM32F411RE  |DESCRIPTION

 VCC        |3.3V         |
 GND        |GND          |
 SCL        |PB8          |Serial clock line
 SDA        |PB9          |Serial data line
 */

/* I2C address */
#ifndef SSD1106_I2C_ADDR
#define SSD1106_I2C_ADDR         0x3C
#endif

/* SSD1106 settings */
/* SSD1106 width in pixels */
#ifndef SSD1106_WIDTH
#define SSD1106_WIDTH            128
#endif
/* SSD1106 LCD height in pixels */
#ifndef SSD1106_HEIGHT
#define SSD1106_HEIGHT           64
#endif

typedef enum {
      SSD1106_COLOR_BLACK = 0x00, /*!< Black color, no pixel */
      SSD1106_COLOR_WHITE = 0x01 /*!< Pixel is set. Color depends on LCD */
} SSD1106_COLOR_t;

/**
 * @brief   A function to initialize the OLED.
 *
 * @param   None
 *
 * @return  Initialization status.
 */
uint8_t SSD1106_init(void);

/**
 * @brief   A function to update the OLED display with the configuration.
 *
 * @param   None
 *
 * @return  None.
 */
void SSD1106_update_screen(void);

/**
 * @brief   A function to fill the OLED display with the specified color.
 *
 * @param   Enumerated value of the color
 *
 * @return  None.
 */
void SSD1106_fill(SSD1106_COLOR_t Color);

/**
 * @brief   A function to draw a pixel at the specified coordinates(x,y) with a
specified color.
 *
 * @param   x     Value of X-coordinate
 *          y     Value of Y-coordinate
 *          color Enumerated value of the color
 *
 * @return  None.
 */
void SSD1106_draw_pixel(uint16_t x, uint16_t y, SSD1106_COLOR_t color);

/**
 * @brief   A function to move the cursor to the specified coordinates(x,y).
 *
 * @param   x     Value of X-coordinate
 *          y     Value of Y-coordinate
 *
 * @return  None.
 */
void SSD1106_gotoXY(uint16_t x, uint16_t y);

/**
 * @brief   A function to display a character on the OLED display.
 *
 * @param   ch    Character to display
 *               Font  Pointer to the font size structure
 *          color Enumerated value of the color
 *
 * @return  The character displayed.
 */
char SSD1106_putc(char ch, FontDef_t *Font, SSD1106_COLOR_t color);

/**
 * @brief   A function to display a string on the OLED display.
 *
 * @param   ch    String to display
 *               Font  Pointer to the font size structure
 *          color Enumerated value of the color
 *
 * @return  Pointer to the string displayed.
 */
char SSD1106_puts(char *str, FontDef_t *Font, SSD1106_COLOR_t color);

/**
 * @brief   A function to clear the entire OLED display.
 *
 * @param   None
 *
 * @return  None.
 */
void SSD1106_clear_screen(void);

/**
 * @brief   A function to clear one line of OLED display.
 *
 * @param   None
 *
 * @return  None.
 */
void SSD1106_clear_line(void);

/**
 * @brief   A function to settle I2C initialization for OLED to STM32
communication.
 *
 * @param   None
 *
 * @return  None.
 */
void SSD1106_i2c_init(void);

/**
 * @brief   A wrapper function to write multiple bytes of data to a specified
memory address of a slave device.
 *
 * @param   saddr Slave address
 *          madr  Memory address
 *          buffer Pointer to the buffer array
 *           length The length of the buffer
 *
 * @return  None.
 */
void SSD1106_i2c_write_multi(uint8_t address, uint8_t reg, char *data,
           uint16_t count);

#endif     /* __OLED_H */
```

9.3.15 fonts.c

```
/******************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya and the University of Colorado are not liable for
* any misuse of this material.
*
******************************************************************************/
/**
* @file   fonts.c
* @brief  A file defining the fonts for the OLED.
*
* @author Krishna Suhagiya
* @date   November 21, 2023
* @reference https://blog.embeddedexpert.io/?p=613
* @revision 1.0
*
*/

#include "fonts.h"

// These fonts will fit into 7x10 pixel size
const uint16_t Font7x10[] = { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
           0x0000, 0x0000, 0x0000,
           0x0000,  // sp
           0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x1000,
0x0000,
           0x0000,  // !
           0x2800, 0x2800, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
           0x0000,  // "
           0x2400, 0x2400, 0x7C00, 0x2400, 0x4800, 0x7C00, 0x4800, 0x4800,
0x0000,
           0x0000,  // #
           0x3800, 0x5400, 0x5000, 0x3800, 0x1400, 0x5400, 0x5400, 0x3800,
0x1000,
           0x0000,  // $
           0x2000, 0x5400, 0x5800, 0x3000, 0x2800, 0x5400, 0x1400, 0x0800,
0x0000,
           0x0000,  // %
           0x1000, 0x2800, 0x2800, 0x1000, 0x3400, 0x4800, 0x4800, 0x3400,
0x0000,
           0x0000,  // &
           0x1000, 0x1000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
           0x0000,  // '
           0x0800, 0x1000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000,
0x1000,
           0x0800,  // (
           0x2000, 0x1000, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800,
0x1000,
           0x2000,  // )
           0x1000, 0x3800, 0x1000, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
           0x0000,  // *
           0x0000, 0x0000, 0x1000, 0x1000, 0x7C00, 0x1000, 0x1000, 0x0000,
0x0000,
           0x0000,  // +
           0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000,
0x1000,
           0x1000,  // ,
           0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x3800, 0x0000, 0x0000,
0x0000,
           0x0000,  // -
           0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000,
0x0000,
           0x0000,  // .
           0x0800, 0x0800, 0x1000, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000,
0x0000,
           0x0000,  // /
           0x3800, 0x4400, 0x4400, 0x5400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000,
           0x0000,  // 0
           0x1000, 0x3000, 0x5000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000,
           0x0000,  // 1
           0x3800, 0x4400, 0x4400, 0x0400, 0x0800, 0x1000, 0x2000, 0x7C00,
0x0000,
           0x0000,  // 2
           0x3800, 0x4400, 0x0400, 0x1800, 0x0400, 0x0400, 0x4400, 0x3800,
0x0000,
           0x0000,  // 3
           0x0800, 0x1800, 0x2800, 0x2800, 0x4800, 0x7C00, 0x0800, 0x0800,
0x0000,
           0x0000,  // 4
           0x7C00, 0x4000, 0x4000, 0x7800, 0x0400, 0x0400, 0x4400, 0x3800,
0x0000,
           0x0000,  // 5
           0x3800, 0x4400, 0x4000, 0x7800, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000,
           0x0000,  // 6
           0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x2000, 0x2000,
0x0000,
           0x0000,  // 7
           0x3800, 0x4400, 0x4400, 0x3800, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000,
           0x0000,  // 8
           0x3800, 0x4400, 0x4400, 0x4400, 0x3C00, 0x0400, 0x4400, 0x3800,
0x0000,
           0x0000,  // 9
           0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000,
0x0000,
           0x0000,  // :
           0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x1000,
0x1000,
           0x1000,  // ;
           0x0000, 0x0000, 0x0C00, 0x3000, 0x4000, 0x3000, 0x0C00, 0x0000,
0x0000,
           0x0000,  // <
           0x0000, 0x0000, 0x0000, 0x7C00, 0x0000, 0x7C00, 0x0000, 0x0000,
0x0000,
           0x0000,  // =
           0x0000, 0x0000, 0x6000, 0x1800, 0x0400, 0x1800, 0x6000, 0x0000,
0x0000,
           0x0000,  // >
           0x3800, 0x4400, 0x0400, 0x0800, 0x1000, 0x1000, 0x0000, 0x1000,
0x0000,
           0x0000,  // ?
           0x3800, 0x4400, 0x4C00, 0x5400, 0x5C00, 0x4000, 0x4000, 0x3800,
0x0000,
           0x0000,  // @
           0x1000, 0x2800, 0x2800, 0x2800, 0x2800, 0x7C00, 0x4400, 0x4400,
0x0000,
           0x0000,  // A
           0x7800, 0x4400, 0x4400, 0x7800, 0x4400, 0x4400, 0x4400, 0x7800,
0x0000,
           0x0000,  // B
           0x3800, 0x4400, 0x4000, 0x4000, 0x4000, 0x4000, 0x4400, 0x3800,
0x0000,
           0x0000,  // C
           0x7000, 0x4800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4800, 0x7000,
0x0000,
           0x0000,  // D
           0x7C00, 0x4000, 0x4000, 0x7C00, 0x4000, 0x4000, 0x4000, 0x7C00,
0x0000,
           0x0000,  // E
           0x7C00, 0x4000, 0x4000, 0x7800, 0x4000, 0x4000, 0x4000, 0x4000,
0x0000,
           0x0000,  // F
           0x3800, 0x4400, 0x4000, 0x4000, 0x5C00, 0x4400, 0x4400, 0x3800,
0x0000,
           0x0000,  // G
           0x4400, 0x4400, 0x4400, 0x7C00, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000,
           0x0000,  // H
           0x3800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x3800,
0x0000,
           0x0000,  // I
           0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x4400, 0x3800,
0x0000,
           0x0000,  // J
           0x4400, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4800, 0x4400,
0x0000,
           0x0000,  // K
           0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x7C00,
0x0000,
           0x0000,  // L
           0x4400, 0x6C00, 0x6C00, 0x5400, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000,
           0x0000,  // M
           0x4400, 0x6400, 0x6400, 0x5400, 0x5400, 0x4C00, 0x4C00, 0x4400,
0x0000,
           0x0000,  // N
           0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000,
           0x0000,  // O
           0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4000, 0x4000, 0x4000,
0x0000,
           0x0000,  // P
           0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x5400, 0x3800,
0x0400,
           0x0000,  // Q
           0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4800, 0x4800, 0x4400,
0x0000,
           0x0000,  // R
           0x3800, 0x4400, 0x4000, 0x3000, 0x0800, 0x0400, 0x4400, 0x3800,
0x0000,
           0x0000,  // S
           0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000,
           0x0000,  // T
           0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000,
           0x0000,  // U
           0x4400, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x1000,
0x0000,
           0x0000,  // V
           0x4400, 0x4400, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800,
0x0000,
           0x0000,  // W
           0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x2800, 0x2800, 0x4400,
0x0000,
           0x0000,  // X
           0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000,
           0x0000,  // Y
           0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x4000, 0x7C00,
0x0000,
           0x0000,  // Z
           0x1800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000,
           0x1800,  // [
           0x2000, 0x2000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800,
0x0000,
           0x0000, /* \ */
           0x3000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000,
           0x3000,  // ]
           0x1000, 0x2800, 0x2800, 0x4400, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
           0x0000,  // ^
           0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
           0xFE00,  // _
           0x2000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000,
           0x0000,  // `
           0x0000, 0x0000, 0x3800, 0x4400, 0x3C00, 0x4400, 0x4C00, 0x3400,
0x0000,
           0x0000,  // a
           0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800,
0x0000,
           0x0000,  // b
           0x0000, 0x0000, 0x3800, 0x4400, 0x4000, 0x4000, 0x4400, 0x3800,
0x0000,
           0x0000,  // c
           0x0400, 0x0400, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0000,
           0x0000,  // d
           0x0000, 0x0000, 0x3800, 0x4400, 0x7C00, 0x4000, 0x4400, 0x3800,
0x0000,
           0x0000,  // e
           0x0C00, 0x1000, 0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000,
           0x0000,  // f
           0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0400,
           0x7800,  // g
           0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000,
           0x0000,  // h
           0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000,
           0x0000,  // i
           0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000,
           0xE000,  // j
           0x4000, 0x4000, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4400,
0x0000,
           0x0000,  // k
           0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000,
           0x0000,  // l
           0x0000, 0x0000, 0x7800, 0x5400, 0x5400, 0x5400, 0x5400, 0x5400,
0x0000,
           0x0000,  // m
           0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000,
           0x0000,  // n
           0x0000, 0x0000, 0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000,
           0x0000,  // o
           0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800,
0x4000,
           0x4000,  // p
           0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0400,
           0x0400,  // q
           0x0000, 0x0000, 0x5800, 0x6400, 0x4000, 0x4000, 0x4000, 0x4000,
0x0000,
           0x0000,  // r
           0x0000, 0x0000, 0x3800, 0x4400, 0x3000, 0x0800, 0x4400, 0x3800,
0x0000,
           0x0000,  // s
           0x2000, 0x2000, 0x7800, 0x2000, 0x2000, 0x2000, 0x2000, 0x1800,
0x0000,
           0x0000,  // t
           0x0000, 0x0000, 0x4400, 0x4400, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0000,
           0x0000,  // u
           0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000,
0x0000,
           0x0000,  // v
           0x0000, 0x0000, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800,
0x0000,
           0x0000,  // w
           0x0000, 0x0000, 0x4400, 0x2800, 0x1000, 0x1000, 0x2800, 0x4400,
0x0000,
           0x0000,  // x
           0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000,
0x1000,
           0x6000,  // y
           0x0000, 0x0000, 0x7C00, 0x0800, 0x1000, 0x2000, 0x4000, 0x7C00,
0x0000,
           0x0000,  // z
           0x1800, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x1000, 0x1000,
0x1000,
           0x1800,  // {
           0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000,
           0x1000,  // |
           0x3000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x1000, 0x1000,
0x1000,
           0x3000,  // }
           0x0000, 0x0000, 0x0000, 0x7400, 0x4C00, 0x0000, 0x0000, 0x0000,
0x0000,
           0x0000,  // ~
           };
```

```
FontDef_t Font_7x10 = { 7, 10, Font7x10 };
```

9.3.16 fonts.h

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file beeper.h
* @brief  A file declaring the fonts for the OLED.
*
* @author Krishna Suhagiya
* @date   November 21, 2023
* @revision 1.0
* @reference https://blog.embeddedexpert.io/?p=613
*
*/

#ifndef __FONTS_H
#define __FONTS_H

#include "stm32f4xx.h"
#include <stdint.h>
#include <string.h>

/**
 * @brief  Font structure used on my LCD libraries
 */
typedef struct {
      uint8_t FontWidth; /*!< Font width in pixels */
      uint8_t FontHeight; /*!< Font height in pixels */
      const uint16_t *data; /*!< Pointer to data font data array */
} FontDef_t;

/**
 * @brief  String length and height
 */
typedef struct {
      uint16_t Length; /*!< String length in units of pixels */
      uint16_t Height; /*!< String height in units of pixels */
} FONTS_SIZE_t;

/**
 * @brief  7 x 10 pixels font size structure
 */
```

```
extern FontDef_t Font_7x10;

#endif     // __FONTS_H
```

9.3.17 i2c.c

```
/*******************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya and the University of Colorado are not liable for
* any misuse of this material.
*
******************************************************************************/
/**
* @file   i2c.c
* @brief  A file defining the I2C communication protocol related APIs for
communication between OLED and STM32.
*
* @author Krishna Suhagiya
* @date   November 21, 2023
* @reference https://blog.embeddedexpert.io/?p=613
* @revision 1.0
*
*/

#include "i2c.h"

void i2c_init(void) {
      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;                     //enable gpiob
clock
      RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;                      //enable i2c1
clock
      GPIOB->MODER |= 0xA0000;                                       //set
pb8and9 to alternative function
      GPIOB->AFR[1] |= 0x44;
      GPIOB->OTYPER |= GPIO_OTYPER_OT8 | GPIO_OTYPER_OT9; //set pb8 and pb9 as
open drain
      I2C1->CR1 = I2C_CR1_SWRST;
      I2C1->CR1 &= ~I2C_CR1_SWRST;
      I2C1->CR2 |= 50;
      I2C1->CCR |= 0x2 | (1 << 15) | (1 << 14);
      I2C1->TRISE = 20;
      //output max rise
      I2C1->CR1 |= I2C_CR1_PE;
}

void i2c_write_byte(char saddr, char maddr, char data) {
      while (I2C1->SR2 & I2C_SR2_BUSY) {
            ;
      }
      //wait until bus not busy
      I2C1->CR1 |= I2C_CR1_START;                             //generate start
      while (!(I2C1->SR1 & I2C_SR1_SB)) {
            ;
      }
      //wait until start bit is set
      I2C1->DR = saddr << 1;                                              //
Send slave address
      while (!(I2C1->SR1 & I2C_SR1_ADDR)) {
            ;
      }
      //wait until address flag is set
      if(I2C1->SR2){
      //clear SR2 by reading it

      }
      while (!(I2C1->SR1 & I2C_SR1_TXE)) {
            ;
      }
      //Wait until Data register empty
      I2C1->DR = maddr;                                                   //
send memory address
      while (!(I2C1->SR1 & I2C_SR1_TXE)) {
            ;
      }
      //wait until data register empty
      I2C1->DR = data;
      while (!(I2C1->SR1 & I2C_SR1_BTF))
            ;
      //wait until transfer finished
      I2C1->CR1 |= I2C_CR1_STOP;                                    //Generate
Stop

}

void i2c_write_multi(char saddr, char maddr, char *buffer, uint8_t length) {

      while (I2C1->SR2 & I2C_SR2_BUSY)
            ;
      //wait until bus not busy
      I2C1->CR1 |= I2C_CR1_START;                             //generate start
      while (!(I2C1->SR1 & I2C_SR1_SB)) {
            ;
      }
      //wait until start is generated
      I2C1->DR = saddr << 1;                                  // Send slave
address
      while (!(I2C1->SR1 & I2C_SR1_ADDR)) {
            ;
      }
      //wait until address flag is set
      if(I2C1->SR2) {
      //Clear SR2
      }
      while (!(I2C1->SR1 & I2C_SR1_TXE))
            ;
      //Wait until Data register empty
      I2C1->DR = maddr;                                                // send memory 
address
      while (!(I2C1->SR1 & I2C_SR1_TXE))
            ;
      //wait until data register empty
      //sending the data
      for (uint8_t i = 0; i < length; i++) {
            I2C1->DR = buffer[i];                                            //filling 
buffer with command or data
            while (!(I2C1->SR1 & I2C_SR1_BTF))
                  ;
      }

      I2C1->CR1 |= I2C_CR1_STOP;                                             //wait 
until transfer finished

}
```

9.3.18 i2c.h

```
/******************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya and the University of Colorado are not liable for
* any misuse of this material.
*
******************************************************************************/
/**
* @file i2c.h
* @brief  A file declaring the I2C communication protocol related APIs for 
communication between OLED and STM32.
*
* @author Krishna Suhagiya
* @date   November 21, 2023
* @revision 1.0
* @reference https://blog.embeddedexpert.io/?p=613
*
*/

#ifndef __I2C_H
#define __I2C_H

#include <stdint.h>
#include "stm32f4xx.h"
#include "delay.h"
```

```
/**
 * @brief   A function to initialize the I2C.
 *
 * @param   None.
 *
 * @return  None.
 */
void i2c_init(void);

/**
 * @brief   A function to write a single byte of data to a specified memory
address of a slave device.
 *
 * @param   saddr Slave address
 *          madr  Memory address
 *          data  Data byte
 *
 * @return  None.
 */
void i2c_write_byte(char saddr, char maddr, char data);

/**
 * @brief   A function to write multiple bytes of data to a specified memory
address of a slave device.
 *
 * @param   saddr Slave address
 *          madr  Memory address
 *          buffer Pointer to the buffer array
 *          length The length of the buffer
 *
 * @return  None.
 */
void i2c_write_multi(char saddr, char maddr, char *buffer, uint8_t length);

#endif     // __I2C_H
```

9.3.19 beeper.c

```
/******************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
******************************************************************************/
/**
* @file   beeper.c
* @brief  A file defining the active buzzer supporting APIs.
```

```
*
* @author Sriya Garde
* @date   November 28, 2023
* @revision 1.0
*
*/

#include "beeper.h"
#include "delay.h"

void beeper_init(void) {
      /* Enable the AHB clock for GPIO port D */
      SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIODEN);

      /* set Port D as output */
      GPIOD->MODER = 0X14;

}

void beeper_enable(void) {
      /* Turn ON the Buzzer */
      GPIOD->BSRR |= GPIO_BSRR_BS2;

      delay(50);

      /* Turn OFF the Buzzer */
      GPIOD->BSRR |= GPIO_BSRR_BR2;
}
```

9.3.20 beeper.h

```
/*****************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file beeper.h
* @brief  A file declaring the active buzzer supporting APIs.
*
* @author Sriya Garde
* @date   November 28, 2023
* @revision 1.0
*
*/

#ifndef __BEEPER_H
#define __BEEPER_H
```

```
#include "stm32f4xx.h"

/**
 * @brief   A function to initialize beeper.
 *
 * @param   NULL
 *
 * @return  NULL
 */
void beeper_init(void);

/**
 * @brief   A function to play the buzzer sound once.
 *
 * @param   NULL
 *
 * @return  NULL
 */
void beeper_enable(void);

#endif     // __BEEPER_H
```

9.3.21 voice.c

```
/*****************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file     voice.c
* @brief    A file defining the APIs for voice recorder and speaker module.
*
* @author   Sriya Garde
* @date     November 25, 2023
* @revision 1.0
*
*/

#include "stm32f4xx.h"
#include "delay.h"
#include "UART.h"

void voice_init() {
      SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIODEN);   // Enable the AHB clock all
GPIO port B

}
```

```
void voice_check() {
      GPIOD->BSRR |= GPIO_BSRR_BS1;                         // Turn ON the Voice
Module

      delay(10);

      GPIOD->BSRR |= GPIO_BSRR_BR1;                         // Turn OFF the Voice
Module
}
```

9.3.22 voice.h

```
/*****************************************************************************
* Copyright (C) 2023 by Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Sriya Garde and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file   voice.h
* @brief  A file declaring the APIs for voice recorder and speaker module.
*
* @author Sriya Garde
* @date   November 25, 2023
* @revision 1.0
*
*/

#ifndef __VOICE_H_
#define __VOICE_H_

/**
 * @brief   A function to initialize voice recorder and speaker module.
 *
 * @param   None
 *
 * @return  None.
 */
void voice_init(void);

/**
 * @brief   A function to output the recorded message.
 *
 * @param   None
 *
 * @return  None.
 */
void voice_check(void);
```

```
#endif /* __VOICE_H_ */
```

9.3.23 security_system_interface.c

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not
liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file   security_system_interface.c
* @brief  A file defining the API to check access based on User IDentifier(UID)
and passwords.
*
* @author Krishna Suhagiya and Sriya Garde
* @date   November 30, 2023
* @revision 1.0
*
*/

#include <stdio.h>
#include "security_system_interface.h"
#include "rfid.h"
#include "oled.h"
#include "keypad.h"
#include "beeper.h"
#include "voice.h"
#include "UART.h"

//Defining fields for checking Valid and Invalid cards
#define VALID_CARDS    2
#define TOTAL_CARDS    4
#define UID_LENGTH     10
#define PASSWORD_LENGTH      5
#define MAX_INPUT_LENGTH     20
uint8_t rfid_id[TOTAL_CARDS] = { 0 };

//Defining char arrays for card UIDs
char *myTags[VALID_CARDS] = { };

char uid_1[UID_LENGTH] = "e39a9fb";
char uid_2[UID_LENGTH] = "23a2a2c5";
char uid_3[UID_LENGTH] = { 0 };
char uid_4[UID_LENGTH] = { 0 };
```

```
bool add_tag = true;
char admin_password[PASSWORD_LENGTH] = "1234";
char security_password[PASSWORD_LENGTH] = "5678";
char received_string[MAX_INPUT_LENGTH];
int i = 0;
unsigned char tagindex = 0;

char buffer[MAX_INPUT_LENGTH];

void check_access(void) {
      //Checking if a card is tapped against the RFID reader
      if (RC522_check_card(rfid_id)) {
      //Extracting the UID of the tapped card.
            sprintf(buffer, "%x%x%x%x", rfid_id[0], rfid_id[1], rfid_id[2],
                        rfid_id[3]);
#ifdef DEBUG
            USART2_string_transmit("\r\n");
#endif
      //Validating the obtained UID of the tapped card against the valid cards
saved in the system
            if ((strcmp(buffer, uid_1) == 0) || (strcmp(buffer, uid_2) == 0)
                        || (strcmp(buffer, uid_3) == 0)
                        || (strcmp(buffer, uid_4) == 0)) {
#ifdef DEBUG
                  USART2_string_transmit("Access Granted \r\n");
#endif
            //Displaying Access Granted on the OLED.
                  SSD1106_gotoXY(0, 0);
                  SSD1106_puts("  Access Granted  ", &Font_7x10, 1);
                  SSD1106_gotoXY(0, 10);
                  SSD1106_clear_line();
                  SSD1106_gotoXY(0, 20);
                  SSD1106_clear_line();
                  SSD1106_update_screen(); //display
                  beeper_enable();
                  delay(50);
            }

            else {
#ifdef DEBUG
                  USART2_string_transmit("Card does not exist.\r\n");
#endif
            //Displaying Card doesnot exist on the LED
                  SSD1106_gotoXY(0, 0);
                  SSD1106_puts("Card doesn't exist", &Font_7x10, 1);
            //Playing Access Denied message on the Playback module
                  voice_check();
#ifdef DEBUG
                  USART2_string_transmit(
                              "Please enter 4 digit admin password for security
pass\r\n");
#endif      //Taking security password input from the user using the Keypad and
displaying on OLED
                  SSD1106_gotoXY(0, 10);
```

```
                 SSD1106_puts("   Please enter   ", &Font_7x10, 1);
                 SSD1106_gotoXY(0, 20);
                 SSD1106_puts("security password:", &Font_7x10, 1);
                 SSD1106_update_screen(); //display
                 strcpy(received_string, check_key());

           //Checking if the correct Security password has been entered and
displaying "Access Granted" if it's correct.
                 if (strcmp(security_password, received_string) == 0) {
                       SSD1106_gotoXY(0, 0);
                       SSD1106_puts("  Access Granted  ", &Font_7x10, 1);
                       SSD1106_gotoXY(0, 10);
                       SSD1106_clear_line();
                       SSD1106_gotoXY(0, 20);
                       SSD1106_clear_line();
                       SSD1106_update_screen(); //display
           //Buzzer ON if access is granted
                       beeper_enable();

           //If the incorrect security password has been entered, get input
displau "Security password wrong" on OLED
                 } else if (strcmp(security_password, received_string) != 0) {
                       SSD1106_gotoXY(0, 0);
                       SSD1106_puts("Security password ", &Font_7x10, 1);
                       SSD1106_gotoXY(0, 10);
                       SSD1106_puts("       wrong      ", &Font_7x10, 1);
                       SSD1106_gotoXY(0, 20);
                       SSD1106_clear_line();
                       SSD1106_update_screen(); //display
                       //Playing Access Denied message on the Playback module
                       voice_check();
                       //Checking if Valid cards in database is equal to total
cards
                       for (unsigned char j = 0; j < VALID_CARDS; j++) {
                             if (strcmp(buffer, myTags[j]) == 0) {
                                   add_tag = false;
                                   break; //// If valid cards = total cards , we
cannot add more cards to the system
                             }
                             add_tag = true; // If valid cards > total cards ,
we can add more cards to the system
                       }

                 //If a card is valid and valid cards < total cards, accept
user input for Admin password to add a card from the Keypad
                       if ((i < VALID_CARDS) && (add_tag == true)) {
#ifdef DEBUG

                 //Displaying "Please enter admin password" on OLED
                             USART2_string_transmit(
                                         "Please enter 4 digit admin password for
adding a card\r\n");
#endif
                             SSD1106_gotoXY(0, 0);
                             SSD1106_puts("  Please enter    ", &Font_7x10, 1);
                             SSD1106_gotoXY(0, 10);
                             SSD1106_puts("  admin password  ", &Font_7x10, 1);
                             SSD1106_gotoXY(0, 20);
                             SSD1106_puts("  to add a card:  ", &Font_7x10, 1);
                             SSD1106_update_screen(); //display

                       //Validate admin password, and if correct, add the card
as a valid card to the system
                             if (strcmp(admin_password, check_key()) == 0) {
                                  myTags[i] = strdup(buffer);
                                  if (i == 0) {
                                        strcpy(uid_3, buffer);
                                  }
                                  if (i == 1) {
                                        strcpy(uid_4, buffer);
                                  }
#ifdef DEBUG

                       //Display "Adding an access card" on OLED
                                  USART2_string_transmit("Adding an access
card\r\n");
#endif
                                  SSD1106_gotoXY(0, 0);
                                  SSD1106_puts("    Card added    ",
&Font_7x10, 1);
                                  SSD1106_gotoXY(0, 10);
                                  SSD1106_clear_line();
                                  SSD1106_gotoXY(0, 20);
                                  SSD1106_clear_line();
                                  SSD1106_update_screen(); //display
#ifdef DEBUG
                                  USART2_string_transmit("\r\n");
#endif
                                  i++;
                       //Checking the valid cards database for a match for the
tapped card
                                  for (unsigned char j = 0; j < VALID_CARDS;
j++) {

                                        if (strcmp(buffer, myTags[j]) == 0) {
                                              tagindex = j;
                                              break;
                                        }
                                  }

                 // Granting access to Valid cards by checking UIDs from system
database.
                 //Giving access to the Valid card by displaying "Access
granted" and beeping buzzer
                                  if (tagindex < VALID_CARDS) {
#ifdef DEBUG
                                        USART2_string_transmit("Access
granted\r\n");
#endif
```

```
                                        SSD1106_gotoXY(0, 0);
                                        SSD1106_puts("  Access Granted  ",
&Font_7x10, 1);
                                        SSD1106_gotoXY(0, 10);
                                        SSD1106_clear_line();
                                        SSD1106_gotoXY(0, 20);
                                        SSD1106_clear_line();
                                        SSD1106_update_screen(); //display
                                        beeper_enable();
                                  } else {
#ifdef DEBUG

                 //Rejecting access to the Invalid card by displaying "Access
Rejected" and playing audio on playback module
                                        USART2_string_transmit("Access
rejected\r\n");
                                        USART2_string_transmit("Please try
again\r\n");
#endif
                                        SSD1106_gotoXY(0, 0);
                                        SSD1106_puts("  Access Denied   ",
&Font_7x10, 1);
                                        SSD1106_gotoXY(0, 10);
                                        SSD1106_puts("     Try again.   ",
&Font_7x10, 1);
                                        SSD1106_gotoXY(0, 20);
                                        SSD1106_clear_line();
                                        SSD1106_update_screen(); //display
                                        voice_check();
                                  }

                       //If Admin password entered is wrong, display "Access
Denied" on OLED and ask user to start process again

                             } else {
#ifdef DEBUG
                                  USART2_string_transmit(
                                              "Admin password wrong.Access
Denied\r\n");
#endif
                                  SSD1106_gotoXY(0, 0);
                                  SSD1106_puts("  Admin password  ",
&Font_7x10, 1);
                                  SSD1106_gotoXY(0, 10);
                                  SSD1106_puts("     wrong.      ",
&Font_7x10, 1);
                                  SSD1106_gotoXY(0, 20);
                                  SSD1106_puts("  Access Denied   ",
&Font_7x10, 1);
                                  SSD1106_update_screen(); //display
                                  voice_check();

                             }
```

```
                        }

                        delay(100);

            // If flow is disrupted at any point, system goes into this loop and
states "Acces Denied", "Try again."
                  } else {
#ifdef DEBUG
                        USART2_string_transmit("Admin password wrong.Access
Denied\r\n");
                        USART2_string_transmit("Please try again\r\n");
#endif
                        SSD1106_gotoXY(0, 0);
                        SSD1106_puts("  Admin password  ", &Font_7x10, 1);
                        SSD1106_gotoXY(0, 10);
                        SSD1106_puts("      wrong.      ", &Font_7x10, 1);
                        SSD1106_gotoXY(0, 20);
                        SSD1106_puts("  Access Denied   ", &Font_7x10, 1);
                        SSD1106_gotoXY(0, 30);
                        SSD1106_puts("    Try again.    ", &Font_7x10, 1);
                        SSD1106_update_screen(); //display
                        voice_check();
                  }
            }
            delay(100);
      }
}
```

9.3.24 security_system_interface.h

```
/*****************************************************************************
* Copyright (C) 2023 by Krishna Suhagiya and Sriya Garde
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Krishna Suhagiya, Sriya Garde and the University of Colorado are not
liable for
* any misuse of this material.
*
*****************************************************************************/
/**
* @file     security_system_interface.h
* @brief    An interface file declaring the API to check access based on User
IDentifier(UID) and passwords.
*
* @author   Krishna Suhagiya and Sriya Garde
* @date     November 30, 2023
* @revision 1.0
*
*/
```

```
#ifndef __SECURITY_SYSTEM_H
#define __SECURITY_SYSTEM_H

#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include "security_system_interface.h"

/**
 * @brief   A function to check the access to the system based on the UID and
passwords.
 *
 * @param   None
 *
 * @return  None.
 */
void check_access(void);

#endif /* __SECURITY_SYSTEM_H */
```

9.4 Appendix - Data Sheets and Application Notes

The data sheets and application notes are available in ‘Datasheets’ folder.