DEPARTMENT OF PHYSICS
UNIVERSITY OF COLOMBO

PH 3032 – Embedded System Laboratory

STUDENT ATTENDANCE SYSTEM

Name :- W.H.Pasindu Chamara
Index No:- s16033
Registration No:- 2021s18849

Table of Contents

1 ABSTRACT

The student attendance system is an automated embedded solution designed to streamline the tracking of student attendance using RFID technology. The system utilizes an AVR microcontroller and integrates a Real-Time Clock (RTC) module, GSM module, LCD display, 1x4 keypad and RFID reader to register and record attendance. Authorized students mark their attendance by scanning RFID cards, triggering an LED, buzzer, and SMS alert to the admin. Additionally, an administrator can manage student registrations via a password-protected mode. The system also features a reset function to clear records. The integration with a mobile app allows for remote management of attendance and communication with parents of absent students.

2 INTRODUCTION

Efficient and accurate attendance tracking is a critical administrative task in educational institutions. As technology advances, there is a growing need for automated solutions that streamline this process, ensuring reliability, accuracy, and ease of use.

This project introduces an embedded system designed to automate student attendance using RFID technology, integrating a GSM module for real-time alerts. At the core of the system is the ATmega328P microcontroller, an 8-bit AVR-based microcontroller that provides the processing power. The ATmega328P features 32K bytes of in system programmable flash memory, 1K bytes of EEPROM, and 2K bytes of internal SRAM, making it ideal for embedded applications that require efficient data handling.

The system utilizes RFID cards to identify students, with each card linked to a unique student ID stored in the system's memory. When a registered student scans their RFID card, the system logs their attendance, blinks a green LED, sounds a buzzer, and sends an SMS notification to the administrator. The system displays prompts and statuses on an LCD screen connected to the microcontroller. A 1x4 keypad is used for secure password entry during critical operations, such as registering new students or resetting the system. In addition to the core attendance function, the system includes a mobile app that allows the admin to manage attendance records, register new students, and notify the parents of absent students.

Normal Mode: Displays the current time and prompts students to scan their RFID cards. Registered students successfully mark their attendance, while unregistered students are notified that they are not in the system.

Registration Mode: Accessed by pressing a dedicated button and entering a password using the 1x4 keypad. The admin can register new students by scanning their RFID cards, with visual and auditory feedback provided to confirm registration.

Reset Mode: Allows the admin to clear all stored data after password entry, ensuring system security and preventing unauthorized data manipulation.

The GSM module enhances the system by sending SMS alerts to the admin whenever a registered student scans their RFID card, providing real time updates. This feature is particularly useful for administrators who want to monitor attendance remotely. Additionally, the mobile application offers flexibility by allowing the admin to manage student data and send notifications to parents.

3 METHODOLOGY

3.1 ATMega328P Pinouts and PCB diagram

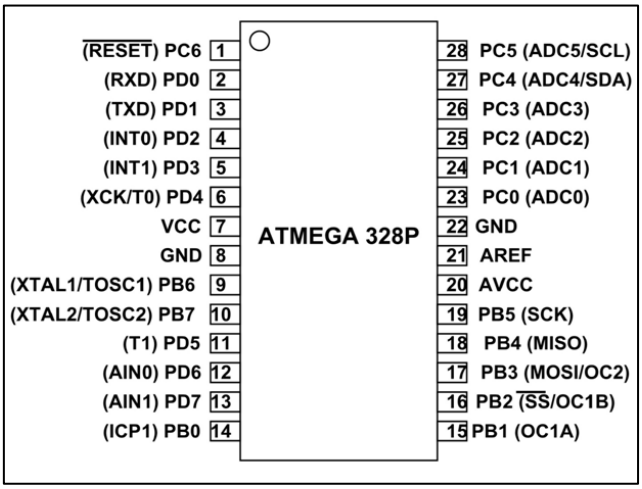

Figure 1: Labeling of pins on the ATMega328P microcontroller.

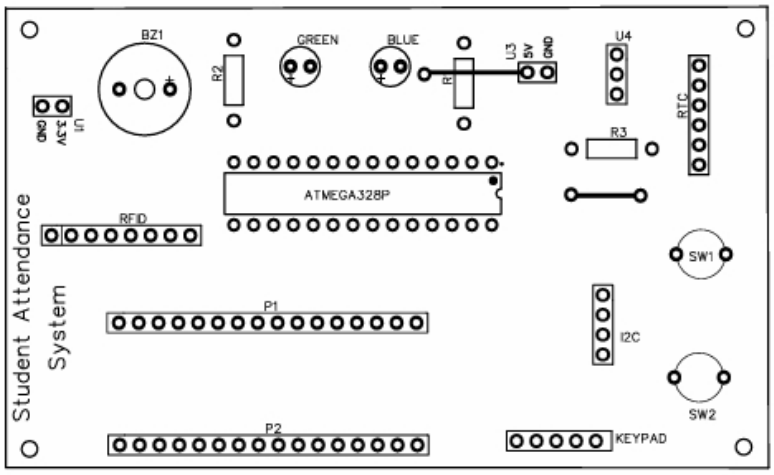

Figure 2: PCB Front Side

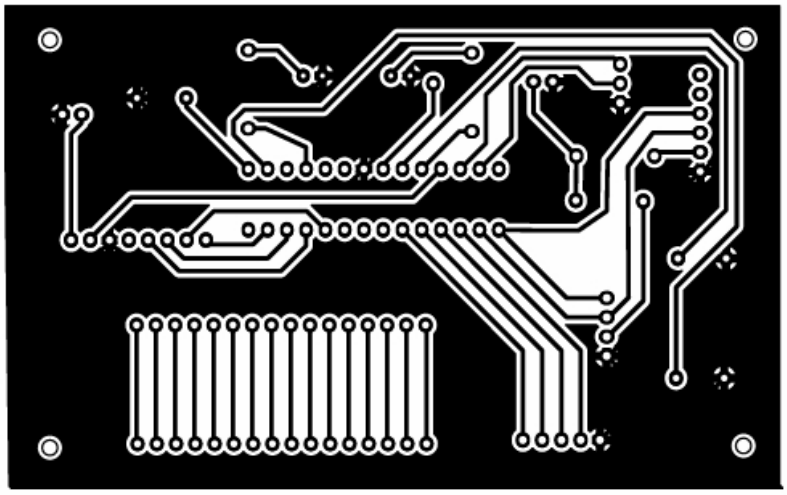

Figure 3: PCB Back Side

3.2 Set the RFID Module

The RFID module allows students to scan their IDs for attendance marking. The communication between the microcontroller and the MFRC522 RFID reader is established using SPI.

- In the **initSPI()** function, the necessary pins for SPI communication are configured. Pins PB3, PB5, and PB2 are set as outputs to handle the MOSI, SCK, and SS lines, respectively.
- The SPCR is configured to enable SPI in master mode with clock rate control.
- Data transmission and reception through SPI are done using the **SPI_send()** and **SPI_receive()** functions. These functions handle writing and reading to and from the MFRC522 module by utilizing the SPI Data Register (SPDR).

```
void initSPI(void) {
   DDRB = (1<<PB3) | (1<<PB5) | (1<<PB2); // Set MOSI, SCK, SS as output
   SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0); // Enable SPI, Master mode, set clock rate
}
```

- **MFRC522_write()**: Sends data to a specified register using the SPI protocol.
- **MFRC522_read()**: Reads data from a specified register by first sending a read command and then retrieving the response from the MFRC522.

3.3 Set the RTC Module

The RTC module is a widely used real-time clock that communicates with microcontrollers via the I2C protocol. It provides highly accurate time and date tracking, maintaining data even during power loss due to its internal battery. The time and date from RTC module and converting the data between Binary Coded Decimal (BCD) and decimal format.

- The function **rtc_read_time()** reads the current time (hours, minutes, and seconds) from the module.
- The values for seconds, minutes, and hours are read sequentially using **i2c_read_ack()** for the first two values (seconds and minutes) and **i2c_read_nack()** for the last value (hours).
- The data is stored in the provided pointer variables seconds, minutes, and hours.

The function **rtc_read_date()** reads the current date information from the module.

The RTC module stores the time and date values in Binary Coded Decimal (BCD) format, which is a compact representation of decimal numbers where each digit is stored as a 4 bit binary number. The microcontroller typically processes values in standard decimal format. Therefore, conversions between BCD and decimal are necessary.

The function **bcd_to_decimal()** converts a BCD value to its equivalent decimal form.

```
uint8_t bcd_to_decimal(uint8_t bcd) {
    return ((bcd / 16 * 10) + (bcd % 16)); // Convert BCD to decimal
}
```

3.4 LCD Display

The LCD display, focusing on the I2C communication protocol and the specific commands required to control the display. The I2C protocol uses a clock line (SCL) and a data line (SDA) for data transfer. The LCD shows prompts such as “Scan the ID,” “Enter Password” and status updates for registered and unregistered students.

I2C Communication

- **i2c_start()**: Initiates the communication by sending a start condition and the device address.
- **i2c_write()**: Sends data to the slave device.
- **i2c_read_ack() / i2c_read_nack()**: Reads data with acknowledgment (ACK) or no acknowledgment (NACK).
- **i2c_stop()**: Terminates the communication.

LCD Initialization

The function **LCD_Init()** initializes the LCD to work in 4-bit mode, enabling the microcontroller to communicate with the LCD using only 4 data lines instead of 8.

The LCD from an unknown state to **4-bit mode**. These commands are transmitted using the **LCD_SendCommand()** function.

- The **0x03** command is sent multiple times to reset the LCD and prepare it for 4-bit mode.
- The **0x02** command is then sent, which officially sets the LCD to 4-bit mode.
- The **0x28** command sets the function parameters (4-bit mode, 2-line display, 5x8 dot matrix).
- The **0x0C** command turns on the display without the cursor or blinking.
- The **0x06** command sets the entry mode (automatic increment, no display shift).
- The **0x01** command is used to clear the display.

Send Command to LCD

The function **LCD_SendCommand()** for sending a command to the LCD. The command is split into two parts

1. The high nibble is sent first, followed by a pulse on the **enable pin** (E) to latch the data.
2. The low nibble is then sent in a similar manner.

```
void LCD_SendCommand(uint8_t cmd) {
  i2c_start(LCD_ADDR<<1);  // Start I2C communication
  i2c_write((cmd & 0xF0) | 0x08); // Send high nibble
  i2c_write((cmd & 0xF0) | 0x0C); // Enable bit high
  i2c_write((cmd & 0xF0) | 0x08); // Enable bit low
  i2c_write((cmd << 4) | 0x08);   // Send low nibble
  i2c_write((cmd << 4) | 0x0C);   // Enable bit high
  i2c_write((cmd << 4) | 0x08);   // Enable bit low
  i2c_stop();  // End I2C communication
}
```

Send Data to LCD

The function **LCD_SendData()** works similarly to **LCD_SendCommand()**, but instead of sending control commands, it sends character data to be displayed on the screen.

```
void LCD_SendData(uint8_t data) {
  i2c_start(LCD_ADDR<<1);  // Start I2C communication
  i2c_write((data & 0xF0) | 0x09); // Send high nibble with RS = 1
  i2c_write((data & 0xF0) | 0x0D); // Enable bit high
  i2c_write((data & 0xF0) | 0x09); // Enable bit low
  i2c_write((data << 4) | 0x09);   // Send low nibble
  i2c_write((data << 4) | 0x0D);   // Enable bit high
  i2c_write((data << 4) | 0x09);   // Enable bit low
  i2c_stop();  // End I2C communication
}
```

Print Data to LCD

The **LCD_Print()** function sends a string of characters to the LCD by repeatedly calling **LCD_SendData()** for each character in the string. It loops through the string, sending each character's ASCII value to the LCD's DDRAM to be displayed.

```
void LCD_Print(char *str) {
  while (*str) {
    LCD_SendData(*str++);
  }
}
```

3.5 1x4 Keypad (Password Entry)

The 1x4 keypad is used to capture numeric input from the admin in the system. It consists of four keys, each corresponding to a specific input.

Keypad Initialization

- **DDR** (Data Direction Register): Set the keypad pins as input.
- Pull-up Resistors: Enable pull-up resistors on each input pin.

```
// Set PORTC pins as input for the 1x4 keypad
DDRC &= ~((1 << PIN_KEY_1) | (1 << PIN_KEY_2) | (1 << PIN_KEY_3) | (1 << PIN_KEY_4));

// Enable pull-up resistors on input pins
PORTC |= (1 << PIN_KEY_1) | (1 << PIN_KEY_2) | (1 << PIN_KEY_3) | (1 << PIN_KEY_4);
```

Debouncing

- This debounceDelay() function is used to filter out signal fluctuations during keypresses.

```
#define DEBOUNCE_TIME 10  // 10 ms debounce time
void debounceDelay() {
   _delay_ms(DEBOUNCE_TIME); // Wait for debounce time to ensure stable keypress
detection
}
```

Key Detection

- The function **getKeyPressed()** scans the input pins to detect if any key is pressed by comparing the current state of the keys with the previous state (lastKeyState).
- A keypress is detected when the previous state is high (1) and the current state is low (0), signifying a transition from unpressed to pressed.

```
int getKeyPressed() {
   uint8_t currentState[KEY_NUM] = {
      (PINC & (1 << PIN_KEY_1)) >> PIN_KEY_1,
      (PINC & (1 << PIN_KEY_2)) >> PIN_KEY_2,
      (PINC & (1 << PIN_KEY_3)) >> PIN_KEY_3,
      (PINC & (1 << PIN_KEY_4)) >> PIN_KEY_4
   };

   for (int i = 0; i < KEY_NUM; i++) {
      if (lastKeyState[i] == 1 && currentState[i] == 0) {
         debounceDelay();  // Debounce
         lastKeyState[i] = currentState[i];  // Update the last state
         return i + 1;  // Return key number (1-based index)
```

```
    }
    lastKeyState[i] = currentState[i];
  }
  return 0;  // No key pressed
}
```

- Once a keypress is detected, the corresponding key number (1-4) is returned. The system then converts this key number into its respective character ('1', '2', '3', or '4') for further processing.
- The admin enters a numeric password by pressing the keys. Each keypress is stored in an array **enteredPassword[]** and displayed on the LCD using the **LCD_SendData()** function.
- The system waits for the admin to input a complete password (determined by PASSWORD_LENGTH), after which the password is processed.

Timeout Action

if **(elapsed_time >= 2000)** Checks if elapsed_time has reached 2 seconds.

- **LCD_SendCommand(0x01)**: Clears the LCD to prepare for a new message.
- **LCD_Print("Press Register")**: Displays a prompt instructing the user to press the "Register" button to exit.
- **LCD_SendCommand(0xC0)** and **LCD_Print("Button to Exit")** After press button go to Normal Mode.

3.6 GSM Module (SMS Communication)

The GSM module sends SMS notifications to the admin whenever a student's attendance is marked. Communication with the GSM module is handled through UART.

The **UART_init()** function configures the UART with the specified baud rate for serial communication with the GSM module. This function calculates the **ubrr** value to set the baud rate accurately by using the formula

$$UBRR = \frac{F_{CPU}}{16 \times baud\ rate} - 1$$

- Transmit and receive functionalities are enabled with the flags **TXEN0** and **RXEN0**, respectively.
- The **UART_send_char()** function transmits a single character by waiting until the UART Data Register Empty (UDRE0) flag indicates that the transmit buffer is empty, then loading the character into the UDR0 register to be sent.
- **UART_send_string()** sends an entire string to the GSM module by calling **UART_send_char()** for each character in the string, facilitating the transmission of multiple AT commands sequentially.

- The **GSM_send_SMS()** function sends an SMS with a predefined attendance message containing an ID number.
- Using a sequence of AT commands, the function initializes the GSM module, configures it for text mode, and specifies the recipient's phone number.
- The **mark_and_send_ID()** function integrates the GSM messaging functionality with the LCD display for user feedback.
- The function displays a message on the LCD to indicate the ID has been marked, followed by a **"Sending SMS"** notification.
- After sending the SMS, a confirmation message **"SMS Sent"** is displayed on the LCD to confirm successful transmission.

3.7 EEPROM

The EEPROM is used to store and retrieve data about RFID cards. Each card has a unique serial number in RFID tag and an assigned ID number. This system includes functions to check if a card is already registered and to store a new card.

- The function **isUnregistered()** verifies if a card's serial number is already registered in the EEPROM. It reads each card entry from EEPROM, comparing the stored **tag_id** with the provided serial. If a match is found, the card is considered registered, and its **id_number** is assigned to **assigned_id**. If no match is found after checking all entries, the function returns and indicating the card is unregistered.
- The **storeCard()** function adds a new card to EEPROM if space is available. It iterates through EEPROM entries, searching for an empty slot (indicated by default values 0xFF in tag_id). When an empty slot is found, an **id_number** is assigned sequentially (e.g., ID_1, ID_2) based on the entry index and the **tag_id** is stored. The assigned ID is displayed on an LCD.

3.8 LED and Buzzer

Normal Mode

- When an RFID card is detected, the green LED (PD5) blinks, and the buzzer (PB0) rings to notify the user that the card is being scanned.

Register Mode

- The blue LED (PD3) blinks, and the buzzer (PB0) rings to notify the admin that a new card is being registered. This behavior provides a clear indication of the system's state, differentiating between normal scanning and registration.

3.9 Buttons

The system has buttons for entering registration and reset modes.

- **Register Button (PD4)**

  Toggles between Normal and Register modes. Each press of this button changes the mode state, allowing easy mode switching for the user.

- **Reset Button (PD7)**

  Activates Reset Mode to clear stored data. When pressed again after clearing memory, it returns the system to Normal Mode.

Implement a debounce delay for stable button detection to avoid rapid mode changes due to button noise.

4 RESULTS AND ANALYSIS

The system successfully integrates multiple functionalities to streamline attendance tracking and improve administrative oversight. In the "Normal" operation mode, registered students scan their RFID tags, upon which the system records their attendance and transmits this information to the admin's mobile application via GSM. This allows the administrator to manage and monitor attendance efficiently in real time. For each attendance entry, the admin can send automated SMS notifications directly to the parents of absent students, enhancing parental engagement and accountability. The "Register" mode enables new student entries to be added securely with password verification, ensuring that only authorized personnel can modify the list of registered students.

Furthermore, for unregistered students, the LCD displays a "Not Registered" message, providing clear and immediate feedback. This feature prevents unauthorized access while allowing administrators to easily identify students who need to be enrolled. During testing, each mode demonstrated consistent functionality, with RFID and GSM integration proving to be both responsive and reliable. The LED and buzzer feedback in both modes offered an additional layer of interaction, helping to maintain user awareness of the current system state. Overall, this integrated design supports a secure and efficient attendance management solution that aligns well with administrative and parental needs.

Figure 4: System at Start

Figure 5: Normal Mode

Figure 6: Not Registered Student

Figure7: When enter Register Mode or Reset Mode should enter password

Figure 8: Register New Student

Figure 9: After finish register student go to Normal Mode New student can mark attendance

Figure 10: Send attendance to Admin via SMS

Figure 11: Press Reset button then enter password go to Reset Mode

Figure 12: Clear all ID's in memory

5 DISCUSSION AND CONCLUSION

The RFID and GSM based attendance system is highlighted, along with its potential for significant improvements in school attendance management and parental involvement. This system not only automated the attendance tracking process but also introduced real time communication with parents, offering a proactive approach to student monitoring. Through each design phase, the integration of RFID, GSM, and LCD displays was validated to be functional and reliable, allowing the administrator to manage and oversee the attendance status remotely. The addition of a password protected registration mode is especially notable, as it enhances security by ensuring that only authorized personnel can add new students to the system. This design prioritizes both user interaction and security, essential features in educational applications.

The dual button setup enabled seamless mode switching, with the "Register" button toggling between normal and registration modes and the "Reset" button allowing memory clearance, thereby preventing data overflow and ensuring a clean restart when needed. The ability of the system to display "Not Registered" for unauthorized tags provides real time feedback, helping to prevent unauthorized entries and guiding users in real time. The system's accuracy in identifying registered versus unregistered tags was instrumental in achieving its purpose, further enhanced by the immediate GSM notification to the administrator, which enables prompt response and attendance updates.

In conclusion, the RFID attendance system presented in this project demonstrates an effective approach to streamlining school attendance management. With further development, such systems could expand to include additional features such as detailed reporting tools, integration with school databases, and enhanced user interfaces for administrators. The successful testing and deployment of this system affirms the feasibility of similar implementations in educational institutions, emphasizing automated, real time, and secure attendance tracking as essential features for modern school environments. The project's outcomes show that this system provides not only operational efficiency but also a positive impact on parental engagement, which is a critical aspect of academic success and student accountability.

6 REFERENCE

- *Microchip Technology Inc., 2015. ATmega328P Automotive Microcontrollers. [online] Available at:* https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

- *Embedded system project: RFID based attendance system - javatpoint* (no date) *www.javatpoint.com*. Available at: https://www.javatpoint.com/embedded-system-project-rfid-based-attendance-system (Accessed: 29 October 2024).

- Admin (2020) *Introduction to DS3231 RTC module " Electroduino*, *ElectroDuino*. Available at: https://www.electroduino.com/ds3231-rtc-module-how-its-works/ (Accessed: 29 October 2024).

- Admin *et al.* (2022) *8051 - GSM interfacing*, *EmbeTronicX*. Available at: https://embetronicx.com/tutorials/microcontrollers/8051/gsm-interfacing-with-8051/ (Accessed: 29 October 2024).

- Magdy, K. *et al.* (2023) *Interfacing I2C LCD 16x2 with pic: MPLAB XC8 mikroc library*, *DeepBlue*. Available at: https://deepbluembedded.com/interfacing-i2c-lcd-16x2-tutorial-with-pic-microcontrollers-mplab-xc8/ (Accessed: 29 October 2024).

- *Command-Line tools : Android studio : Android developers* (no date) *Android Developers*. Available at: https://developer.android.com/tools (Accessed: 29 October 2024).

7 APPENDIX

https://drive.google.com/file/d/11QDS_IxuFwUJaq4mPFkVe6w0stXTiL2H/view?usp=sharing

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/eeprom.h>
#include <string.h>
#include <stdarg.h>
#include <stdio.h>

// Define CPU clock and LCD I2C address
#define F_CPU 16000000UL // Define CPU clock as 16 MHz
#define LCD_ADDR 0x27    // LCD I2C address
#define KEY_NUM 4
#define PASSWORD_LENGTH 4

#define PIN_KEY_1 PC0
#define PIN_KEY_2 PC1
#define PIN_KEY_3 PC2
#define PIN_KEY_4 PC3

// Pins for the LEDs and debug
#define LED_PIN_PD3 PD3  // LED connected to PD3
//#define DEBUG_PIN PD4
#define LED_PIN_PD5 PD5  // LED connected to PD3
#define LED_PIN_PD7 PD7
#define LED_PIN_PB0 PB0  // Additional LED connected to PB0
#define BUTTON_PIN PD4
#define RESET_PIN PD7
#define DEBOUNCE_TIME 10  // 10 ms debounce time
#define MAX_LEN 16

// Status codes
#define MI_OK                 0
#define MI_NOTAGERR           1
#define MI_ERR                2

// MFRC522 command set
#define PCD_IDLE              0x00
#define PCD_AUTHENT           0x0E
#define PCD_RECEIVE           0x08
#define PCD_TRANSMIT          0x04
#define PCD_TRANSCEIVE        0x0C
#define PCD_RESETPHASE        0x0F
#define PCD_CALCCRC           0x03

// Mifare_One card command set
#define PICC_REQIDL           0x26
#define PICC_REQALL           0x52
#define PICC_ANTICOLL         0x93

// MFRC522 registers
#define CommandReg            0x01
#define CommIEnReg            0x02
#define CommIrqReg            0x04
#define ErrorReg              0x06
#define Status1Reg            0x07
#define Status2Reg            0x08
#define FIFODataReg           0x09
#define FIFOLevelReg          0x0A
#define ControlReg            0x0C
#define BitFramingReg         0x0D
#define ModeReg               0x11
```

```
#define TxModeReg             0x12
#define RxModeReg             0x13
#define TxControlReg          0x14
#define TxASKReg              0x15
#define TModeReg              0x2A
#define TPrescalerReg         0x2B
#define TReloadRegH           0x2C
#define TReloadRegL           0x2D

// Structure to hold RFID card ID and tag ID
typedef struct {
    uint8_t id_number;
    uint8_t tag_id[5];
} RFIDCard;

uint8_t lastKeyState[KEY_NUM] = {1, 1, 1, 1};  // 1 = not pressed, 0 = pressed
// Password checking
const char correctPassword[PASSWORD_LENGTH + 1] = "2431";  // Correct password
char enteredPassword[PASSWORD_LENGTH + 1];  // Store entered password as a string
uint8_t enteredIndex = 0;  // Track entered digits

// Function prototypes
void initSPI(void);
void SPI_send(char data);
char SPI_receive(void);
void initLEDs(void);
void blinkLED(uint8_t led_pin);
void LCD_Init(void);
void LCD_SendCommand(uint8_t cmd);
void LCD_SendData(uint8_t data);
void LCD_Print(char *str);
void MFRC522_init(void);
void MFRC522_reset(void);
void MFRC522_write(uint8_t addr, uint8_t val);
uint8_t MFRC522_read(uint8_t addr);
void MFRC522_setBitMask(uint8_t reg, uint8_t mask);
void MFRC522_clearBitMask(uint8_t reg, uint8_t mask);
void MFRC522_antennaOn(void);
uint8_t MFRC522_request(uint8_t reqMode, uint8_t *TagType);
uint8_t MFRC522_toCard(uint8_t command, uint8_t *sendData, uint8_t sendLen, uint8_t
*backData, uint8_t *backLen);
uint8_t MFRC522_anticoll(uint8_t *serNum);
uint8_t isUnregistered(uint8_t *serial, uint8_t *assigned_id);
void storeCard(uint8_t *serial);
void i2c_init(void);
void i2c_start(uint8_t address);
void i2c_stop(void);
void i2c_write(uint8_t data);
uint8_t i2c_read_ack(void);
uint8_t i2c_read_nack(void);
void rtc_read_time(uint8_t *hours, uint8_t *minutes, uint8_t *seconds);
void rtc_read_date(uint8_t *day, uint8_t *date, uint8_t *month, uint8_t *year);
uint8_t bcd_to_decimal(uint8_t bcd);
uint8_t decimal_to_bcd(uint8_t decimal);
void LCD_Printf(const char *fmt, ...);
void GSM_send_SMS(char *id_number);
void mark_and_send_ID(uint8_t assigned_id);
void clearStoredCards(void);
void debounceDelay(void);
int getKeyPressed(void);
void checkPassword(void);
void resetPasswordEntry(void);

// DS3231 RTC address
```

```
#define DS3231_ADDRESS 0x68

void UART_init(unsigned int baudrate) {
    unsigned int ubrr = F_CPU / 16 / baudrate - 1;
    UBRR0H = (unsigned char)(ubrr >> 8); // Set baud rate
    UBRR0L = (unsigned char)ubrr;
    UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Enable transmitter and receiver
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // Set frame: 8data, 1 stop bit
}
// Send a single character via UART
void UART_send_char(char data) {
    while (!(UCSR0A & (1 << UDRE0))); // Wait until buffer is empty
    UDR0 = data; // Send data
}
void UART_send_string(const char *str) {
    while (*str) {
        UART_send_char(*str++);
    }
}

// Function to send SMS with the ID number
void GSM_send_SMS(char *id_number) {
    UART_send_string("AT\r"); // Initialize GSM module
    _delay_ms(500);

    UART_send_string("AT+CMGF=1\r"); // Set SMS to text mode
    _delay_ms(500);

    UART_send_string("AT+CMGS=\"+94771590486\"\r"); // Set the recipient phone number
    _delay_ms(500);

    UART_send_string(id_number); // Send the ID number in the SMS
    UART_send_char(26); // ASCII code for CTRL+Z to send SMS
    _delay_ms(500);
}

void clearStoredCards() {
    RFIDCard card;
    memset(&card, 0xFF, sizeof(RFIDCard));  // Set all bytes to 0xFF (empty EEPROM
state)

    for (uint8_t i = 0; i < 10; i++) {
        eeprom_write_block((const void*)&card, (void*)(i * sizeof(RFIDCard)),
sizeof(RFIDCard));
        _delay_ms(10);  // Delay to ensure stability after each EEPROM write
    }

    // Display reset confirmation on the LCD
    LCD_SendCommand(0x01);  // Clear the LCD
    LCD_Print("Clear All..");
    _delay_ms(200);  // Give the user time to see the message
}

// Debounce delay function
void debounceDelay() {
    _delay_ms(DEBOUNCE_TIME);
}

void resetPasswordEntry() {
    enteredIndex = 0;
    enteredPassword[0] = '\0';  // Clear the entered password
    LCD_SendCommand(0x01); // Clear display
    LCD_SendCommand(0x80); // Set cursor to the beginning of first line
    LCD_Print("Enter Password:");
```

```
    LCD_SendCommand(0xC0); // Move cursor to second line
}

uint8_t registerMode = 0;
// Keypad functions
int getKeyPressed() {
    uint8_t currentState[KEY_NUM] = {
        (PINC & (1 << PIN_KEY_1)) >> PIN_KEY_1,
        (PINC & (1 << PIN_KEY_2)) >> PIN_KEY_2,
        (PINC & (1 << PIN_KEY_3)) >> PIN_KEY_3,
        (PINC & (1 << PIN_KEY_4)) >> PIN_KEY_4
    };

    for (int i = 0; i < KEY_NUM; i++) {
        if (lastKeyState[i] == 1 && currentState[i] == 0) {
            debounceDelay();  // Debounce
            lastKeyState[i] = currentState[i];  // Update the last state
            return i + 1;  // Return key number (1-based index)
        }
        lastKeyState[i] = currentState[i];
    }
    return 0;  // No key pressed
}

int main(void) {
    uint8_t status;
    uint8_t str[MAX_LEN];
    uint8_t buttonPressed = 0;
    uint8_t hours, minutes, seconds;
    uint8_t day, date, month, year;
    uint8_t assigned_id;
    uint16_t elapsed_time = 0;

    initSPI();
    initLEDs(); // Initialize both LEDs
    MFRC522_init();
    i2c_init();  // Initialize I2C communication for LCD
    LCD_Init();
    UART_init(9600); // Initialize UART with baud rate 9600
    _delay_ms(500); // Shorter startup delay for the GSM module

    // Display the welcome message in the center of the LCD
    LCD_SendCommand(0x01);  // Clear the LCD

    // First line: "Attendance" centered (10 characters, start at position 3 for 16x2
LCD)
    LCD_SendCommand(0x83);  // Set cursor to the 4th position (index starts from 0)
    LCD_Print("Attendance");

    // Second line: "System" centered (6 characters, start at position 5 for 16x2
LCD)
    LCD_SendCommand(0xC5);  // Set cursor to the 6th position of the second line
    LCD_Print("System");

    _delay_ms(100);  // Keep the message on screen for 2 seconds

   //DDRD |= (1 << DEBUG_PIN); // Set DEBUG_PIN as output
   // Set PORTC pins as input
    DDRC &= ~((1 << PIN_KEY_1) | (1 << PIN_KEY_2) | (1 << PIN_KEY_3) | (1 <<
PIN_KEY_4));

    // Enable pull-up resistors on input pins
    PORTC |= (1 << PIN_KEY_1) | (1 << PIN_KEY_2) | (1 << PIN_KEY_3) | (1 <<
PIN_KEY_4);
```

```
    while (1) {
        if (buttonPressed == 0) { // Only update RTC when RFID session is not active
            // Continuously read and display the current time and date
            rtc_read_time(&hours, &minutes, &seconds);
            rtc_read_date(&day, &date, &month, &year);

            hours = bcd_to_decimal(hours);
            minutes = bcd_to_decimal(minutes);
            seconds = bcd_to_decimal(seconds);
            date = bcd_to_decimal(date);
            month = bcd_to_decimal(month);
            year = bcd_to_decimal(year);

            LCD_SendCommand(0x01);  // Clear the LCD
            LCD_Printf("Time: %02d:%02d:%02d", hours, minutes, seconds);
            LCD_SendCommand(0xC0); // Move to the second line
            LCD_Printf("Date: %02d/%02d/20%02d", date, month, year);

            _delay_ms(100); // Update every second
            LCD_SendCommand(0x01);  // Clear the LCD
            LCD_Print("Scan the ID");
            _delay_ms(200);
            status = MFRC522_request(PICC_REQIDL, str);
            if (status == MI_OK) {
                status = MFRC522_anticoll(str);
                if (status == MI_OK) {
                    LCD_SendCommand(0x01);  // Clear the LCD
                    LCD_Print("Scanning...");

                    // Blink both LEDs when a card is detected
                    blinkLED(LED_PIN_PD5);
                    blinkLED(LED_PIN_PB0);

                    if (isUnregistered(str, &assigned_id)) {
                        LCD_SendCommand(0x01);  // Clear the LCD
                        LCD_Print("Not Registered");
                        _delay_ms(200);
                    } else {
                       mark_and_send_ID(assigned_id);
                    }
                    LCD_SendCommand(0x01);  // Clear the LCD
                    LCD_Printf("Time: %02d:%02d:%02d", hours, minutes, seconds);
                    LCD_SendCommand(0xC0); // Move to the second line
                    LCD_Printf("Date: %02d/%02d/20%02d", date, month, year);
                }
            }
        }
         while (!(PIND & (1 << BUTTON_PIN))) {  // If the button is pressed (low)
              if (!buttonPressed) {  // First time the button is pressed
                  buttonPressed = 1;  // Mark the button as pressed
                  resetPasswordEntry();  // Reset the password entry process
                  enteredIndex = 0;

                  int key = 0;
                  while (enteredIndex < PASSWORD_LENGTH) {
                      _delay_ms(10);  // Poll every 50ms
                      elapsed_time += 50;
                      key = getKeyPressed();  // Check if a key is pressed

                      if (key) {
                          char keyChar = key + '0';  // Convert the key number to
character
                          LCD_SendData(keyChar);  // Show the key on the LCD
```

```
                        enteredPassword[enteredIndex++] = keyChar;  // Store the
key

                        _delay_ms(DEBOUNCE_TIME);  // Debounce delay to prevent
rapid keypress detection
                        elapsed_time = 0;

                    }
                    if (elapsed_time >= 2000) {
                      LCD_SendCommand(0x01);  // Clear LCD
                      LCD_Print("Press Register");
                      LCD_SendCommand(0xC0);
                      LCD_Print("Button to Exit");

                      // Check for button release to go back to normal mode
                      if ((PIND & (1 << BUTTON_PIN)) && (PIND & (1 << RESET_PIN)))
{
                          buttonPressed = 0;  // Exit reset mode
                          break;  // Exit the password entry loop
                      }
                  }
                }

            enteredPassword[enteredIndex] = '\0';  // Null-terminate the entered
password

            // Check if the entered password matches the correct password
            if (strcmp(enteredPassword, correctPassword) == 0) {
                LCD_SendCommand(0x01);  // Clear the LCD
                LCD_Print("Register Mode");
                LCD_SendCommand(0xC0);
                LCD_Print("Scan the ID");
              }else {
                  // Password was incorrect
                  LCD_SendCommand(0x01);  // Clear the display
                  LCD_SendCommand(0x80);  // Set cursor to the beginning of first
line
                  LCD_Print("Try Again!");
                  _delay_ms(300);  // Show "Try Again!" for 2 seconds
                  resetPasswordEntry();  // Reset for next attempt
                  enteredIndex = 0;  // Reset index
                  buttonPressed = 0;  // Allow button press to trigger again
                  LCD_SendCommand(0x01);  // Clear the LCD
                  LCD_Print("");  // Prompt for a new password
                  _delay_ms(100);

              }
              }
               status = MFRC522_request(PICC_REQIDL, str);
               if (status == MI_OK) {
                  status = MFRC522_anticoll(str);
                  if (status == MI_OK) {
                      LCD_SendCommand(0x01);  // Clear the LCD
                      LCD_Print("Scanning...");

                      // Blink both LEDs when a card is detected
                      blinkLED(LED_PIN_PD3);
                      blinkLED(LED_PIN_PB0);

                      if (isUnregistered(str, &assigned_id)) {
                          // Store the card in EEPROM
                          storeCard(str);
                      } else {
                          // If card is already registered, show its ID
```

```
                        LCD_SendCommand(0x01);  // Clear the LCD
                        LCD_Printf("Already ID_%d", assigned_id);
                        _delay_ms(200);  // Delay to show the message
                    }
                    LCD_SendCommand(0x01);  // Clear the LCD
                   LCD_Print("Register Mode");
                   LCD_SendCommand(0xC0);
                   LCD_Print("Scan the ID");
               }
           }
             }


          if (!(PIND & (1 << RESET_PIN))) {  // Check if RESET_PIN is pressed (low)
               debounceDelay();
               buttonPressed = 1;  // Mark the button as pressed

               // Start by showing "Enter Password"
               resetPasswordEntry();

               int key = 0;
               while (enteredIndex < PASSWORD_LENGTH) {
                   _delay_ms(10);  // Poll every 50ms
                   elapsed_time += 50;
                   key = getKeyPressed();  // Check if a key is pressed

                   if (key) {
                       char keyChar = key + '0';  // Convert the key number to
character
                       LCD_SendData(keyChar);  // Show the key on the LCD
                       enteredPassword[enteredIndex++] = keyChar;  // Store the key

                       _delay_ms(DEBOUNCE_TIME);  // Prevent rapid keypress
detection
                       // Reset the timeout
                       elapsed_time = 0;
                   }
                    // If no key is pressed within 500ms, show "press to Exit"
message
                   if (elapsed_time >= 2000) {
                       LCD_SendCommand(0x01);  // Clear LCD
                       LCD_Print("Press REST");
                       LCD_SendCommand(0xC0);  // Set cursor to first line
                       LCD_Print("Button to Exit");

                       // Check for button release to go back to normal mode
                       if ((PIND & (1 << BUTTON_PIN)) && (PIND & (1 << RESET_PIN)))
{
                           buttonPressed = 0;  // Exit reset mode
                           break;  // Exit the password entry loop
                       }
                   }
               }

               enteredPassword[enteredIndex] = '\0';  // Null-terminate the entered
password

               // Check if the entered password matches the correct one
               if (strcmp(enteredPassword, correctPassword) == 0) {
                   // Clear stored cards
                   LCD_SendCommand(0x01);  // Clear the LCD
                   LCD_Print("RESET Mode");
                   _delay_ms(500);  // Delay for user to see the message
```

```
                    clearStoredCards();  // Execute clear operation

                    // After clearing, reset the state
                    _delay_ms(1000);  // Give time for the EEPROM reset process
                    LCD_SendCommand(0x01);  // Clear the LCD
                    LCD_Print("Press Button to");
                    LCD_SendCommand(0xC0);  // Move to the second line
                    LCD_Print("Exit!!");
                    // Wait until the button is released
                    while (!(PIND & (1 << RESET_PIN))) {
                        // Do nothing, just wait for the button release
                    }
                } else {
                    // Password was incorrect
                    LCD_SendCommand(0x01);  // Clear the display
                    LCD_SendCommand(0x80);  // Set cursor to the beginning of first
line
                    LCD_Print("Try Again!");
                    _delay_ms(300);  // Show "Try Again!" for 2 seconds
                    resetPasswordEntry();  // Reset for next attempt
                }
            }

        if (buttonPressed) { // If the RFID session just ended
            buttonPressed = 0;
            // After the RFID session ends, resume RTC display
            LCD_SendCommand(0x01);  // Clear the LCD

            // Read and display the current time and date
            rtc_read_time(&hours, &minutes, &seconds);
            rtc_read_date(&day, &date, &month, &year);

            hours = bcd_to_decimal(hours);
            minutes = bcd_to_decimal(minutes);
            seconds = bcd_to_decimal(seconds);
            date = bcd_to_decimal(date);
            month = bcd_to_decimal(month);
            year = bcd_to_decimal(year);

            LCD_Printf("Time: %02d:%02d:%02d", hours, minutes, seconds);
            LCD_SendCommand(0xC0); // Move to the second line
            LCD_Printf("Date: %02d/%02d/20%02d", date, month, year);
        }

      }
      return 0;
}
void mark_and_send_ID(uint8_t assigned_id) {
    // Display the ID on the LCD
    LCD_SendCommand(0x01);  // Clear the LCD
    LCD_Printf("ID_%d Marked", assigned_id);
    _delay_ms(200);  // Delay to show the message

     // Show "Sending SMS" on the LCD
    LCD_SendCommand(0x01);  // Clear the LCD
    LCD_Print("Sending SMS");
    _delay_ms(500);  // Give the user a little time to see the message

    // Send SMS with the marked ID
    char id_message[16];
    sprintf(id_message, "ID_%d", assigned_id); // Prepare ID message
    GSM_send_SMS(id_message); // Send the ID via SMS

     // After SMS is sent, show confirmation
```

```
    LCD_SendCommand(0x01);  // Clear the LCD
    LCD_Print("SMS Sent");
    _delay_ms(100);  // Show the confirmation message for 1 second
}

void initSPI(void) {
    DDRB = (1<<PB3) | (1<<PB5) | (1<<PB2);
    SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0);
}

void SPI_send(char data) {
    SPDR = data;
    while(!(SPSR & (1<<SPIF)));
}

char SPI_receive(void) {
    SPI_send(0x00);
    while(!(SPSR & (1<<SPIF)));
    return SPDR;
}

void MFRC522_init(void) {
    MFRC522_reset();
    MFRC522_write(TModeReg, 0x8D);
    MFRC522_write(TPrescalerReg, 0x3E);
    MFRC522_write(TReloadRegL, 30);
    MFRC522_write(TReloadRegH, 0);
    MFRC522_write(TxASKReg, 0x40);
    MFRC522_write(ModeReg, 0x3D);
    MFRC522_antennaOn();
}

void MFRC522_reset(void) {
    MFRC522_write(CommandReg, PCD_RESETPHASE);
}

void MFRC522_write(uint8_t addr, uint8_t val) {
    PORTB &= ~(1<<PB2);
    SPI_send((addr<<1)&0x7E);
    SPI_send(val);
    PORTB |= (1<<PB2);
    _delay_ms(1); // Small delay after each SPI transaction to ensure stability
}

uint8_t MFRC522_read(uint8_t addr) {
    uint8_t val;
    PORTB &= ~(1<<PB2);
    SPI_send(((addr<<1)&0x7E) | 0x80);
    val = SPI_receive();
    PORTB |= (1<<PB2);
    _delay_ms(1); // Small delay after each SPI transaction to ensure stability
    return val;
}

void MFRC522_setBitMask(uint8_t reg, uint8_t mask) {
    uint8_t tmp = MFRC522_read(reg);
    MFRC522_write(reg, tmp | mask);
}

void MFRC522_clearBitMask(uint8_t reg, uint8_t mask) {
    uint8_t tmp = MFRC522_read(reg);
    MFRC522_write(reg, tmp & (~mask));
}
```

```
void MFRC522_antennaOn(void) {
    uint8_t temp = MFRC522_read(TxControlReg);
    if (!(temp & 0x03)) {
        MFRC522_setBitMask(TxControlReg, 0x03);
    }
}

uint8_t MFRC522_request(uint8_t reqMode, uint8_t *TagType) {
    uint8_t status;
    uint8_t backBits;

    MFRC522_write(BitFramingReg, 0x07);
    TagType[0] = reqMode;
    status = MFRC522_toCard(PCD_TRANSCEIVE, TagType, 1, TagType, &backBits);

    if ((status != MI_OK) || (backBits != 0x10)) {
        status = MI_ERR;
    }

    return status;
}

uint8_t MFRC522_toCard(uint8_t command, uint8_t *sendData, uint8_t sendLen, uint8_t
*backData, uint8_t *backLen) {
    uint8_t status = MI_ERR;
    uint8_t irqEn = 0x00;
    uint8_t waitIRq = 0x00;
    uint8_t lastBits;
    uint8_t n;
    uint16_t i;  // Use uint16_t to avoid truncation issues

    switch (command) {
        case PCD_AUTHENT:
            irqEn = 0x12;
            waitIRq = 0x10;
            break;
        case PCD_TRANSCEIVE:
            irqEn = 0x77;
            waitIRq = 0x30;
            break;
        default:
            break;
    }

    MFRC522_write(CommIEnReg, irqEn | 0x80);
    MFRC522_clearBitMask(CommIrqReg, 0x80);
    MFRC522_setBitMask(FIFOLevelReg, 0x80);

    MFRC522_write(CommandReg, PCD_IDLE);

    for (i = 0; i < sendLen; i++) {
        MFRC522_write(FIFODataReg, sendData[i]);
    }

    MFRC522_write(CommandReg, command);
    if (command == PCD_TRANSCEIVE) {
        MFRC522_setBitMask(BitFramingReg, 0x80);
    }

    i = 2000;  // Use a higher limit to avoid timeout issues
    do {
        n = MFRC522_read(CommIrqReg);
        i--;
    } while ((i != 0) && !(n & 0x01) && !(n & waitIRq));
```

```
    MFRC522_clearBitMask(BitFramingReg, 0x80);

    if (i != 0) {
        if (!(MFRC522_read(ErrorReg) & 0x1B)) {
            status = MI_OK;
            if (n & irqEn & 0x01) {
                status = MI_NOTAGERR;
            }

            if (command == PCD_TRANSCEIVE) {
                n = MFRC522_read(FIFOLevelReg);
                lastBits = MFRC522_read(ControlReg) & 0x07;
                if (lastBits) {
                    *backLen = (n - 1) * 8 + lastBits;
                } else {
                    *backLen = n * 8;
                }

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

                for (i = 0; i < n; i++) {
                    backData[i] = MFRC522_read(FIFODataReg);
                }
            }
        } else {
            status = MI_ERR;
        }
    }

    return status;
}

uint8_t MFRC522_anticoll(uint8_t *serNum) {
    uint8_t status;
    uint8_t i;
    uint8_t serNumCheck = 0;
    uint8_t unLen;

    MFRC522_write(BitFramingReg, 0x00);
    serNum[0] = PICC_ANTICOLL;
    serNum[1] = 0x20;
    status = MFRC522_toCard(PCD_TRANSCEIVE, serNum, 2, serNum, &unLen);

    if (status == MI_OK) {
        for (i = 0; i < 4; i++) {
            serNumCheck ^= serNum[i];
        }
        if (serNumCheck != serNum[i]) {
            status = MI_ERR;
        }
    }

    return status;
}

void initLEDs(void) {
    // Initialize both LEDs
    DDRD |= (1<<LED_PIN_PD3);
```

```
    DDRD |= (1<<LED_PIN_PD5);
    DDRB |= (1<<LED_PIN_PB0);
    // Set BUTTON_PIN and RESET_PIN as input and enable internal pull-up resistor
    DDRD &= ~(1 << BUTTON_PIN);  // Set BUTTON_PIN as input
    PORTD |= (1 << BUTTON_PIN);  // Enable pull-up resistor for BUTTON_PIN

    DDRD &= ~(1 << RESET_PIN);   // Set RESET_PIN as input
    PORTD |= (1 << RESET_PIN);   // Enable pull-up resistor for RESET_PIN

}

void blinkLED(uint8_t led_pin) {
    // Blink the specified LED
    if (led_pin == LED_PIN_PD3) {
        PORTD |= (1<<led_pin);
        _delay_ms(50);
        PORTD &= ~(1<<led_pin);
    } else if (led_pin == LED_PIN_PB0) {
        PORTB |= (1<<led_pin);
        _delay_ms(50);
        PORTB &= ~(1<<led_pin);
    } else if (led_pin == LED_PIN_PD5) {
        PORTD |= (1<<led_pin);
        _delay_ms(50);
        PORTD &= ~(1<<led_pin);
    }else if (led_pin == LED_PIN_PD7) {
        PORTD |= (1<<led_pin);
        _delay_ms(50);
        PORTD &= ~(1<<led_pin);
    }
}

uint8_t isUnregistered(uint8_t *serial, uint8_t *assigned_id) {
    RFIDCard card;
    for (uint8_t i = 0; i < 15; i++) {
        eeprom_read_block((void*)&card, (const void*)(i * sizeof(RFIDCard)),
sizeof(RFIDCard));
        _delay_ms(10);  // Add delay after each EEPROM read to ensure stability
        if (memcmp(serial, card.tag_id, 5) == 0) {
            *assigned_id = card.id_number;  // Return the existing ID number
            return 0; // Card is registered
        }
    }
    return 1; // Card is unregistered
}

void storeCard(uint8_t *serial) {
    RFIDCard card;
    for (uint8_t i = 0; i < 15; i++) {
        eeprom_read_block((void*)&card, (const void*)(i * sizeof(RFIDCard)),
sizeof(RFIDCard));
        _delay_ms(10);  // Add delay after each EEPROM read to ensure stability
        if (card.tag_id[0] == 0xFF && card.tag_id[1] == 0xFF) { // EEPROM is empty
(0xFF)
            card.id_number = i + 1;  // Assign ID_1, ID_2, ..., ID_10
            memcpy(card.tag_id, serial, 5);
            eeprom_write_block((const void*)&card, (void*)(i * sizeof(RFIDCard)),
sizeof(RFIDCard));
            _delay_ms(10);  // Add delay after each EEPROM write to ensure stability
            // Display the assigned ID on the LCD
            LCD_SendCommand(0x01);  // Clear the LCD
            LCD_Printf("ID_%d Registered!", card.id_number);
            _delay_ms(200);  // Delay to show the message
            break;
```

```
        }
    }
}

void i2c_init(void) {
    // Initialize I2C (TWI) interface
    TWSR = 0x00; // Prescaler value of 1
    TWBR = ((F_CPU/100000UL) - 16) / 2; // SCL frequency 100kHz
}

void i2c_start(uint8_t address) {
    // Send start condition
    TWCR = (1<<TWSTA) | (1<<TWEN) | (1<<TWINT);
    while (!(TWCR & (1<<TWINT))); // Wait for start to be transmitted

    // Load slave address into data register
    TWDR = address;
    TWCR = (1<<TWEN) | (1<<TWINT); // Clear TWINT to start transmission
    while (!(TWCR & (1<<TWINT))); // Wait for end of transmission
}

void i2c_stop(void) {
    // Send stop condition
    TWCR = (1<<TWSTO) | (1<<TWEN) | (1<<TWINT);
    while (TWCR & (1<<TWSTO)); // Wait for stop to be transmitted
}

void i2c_write(uint8_t data) {
    TWDR = data;
    TWCR = (1<<TWEN) | (1<<TWINT);
    while (!(TWCR & (1<<TWINT))); // Wait for end of transmission
}

uint8_t i2c_read_ack(void) {
    TWCR = (1<<TWEN) | (1<<TWINT) | (1<<TWEA);
    while (!(TWCR & (1<<TWINT)));
    return TWDR;
}

uint8_t i2c_read_nack(void) {
    TWCR = (1<<TWEN) | (1<<TWINT);
    while (!(TWCR & (1<<TWINT)));
    return TWDR;
}

void LCD_Init(void) {
    // Initialize LCD (Assuming 4-bit mode)
    _delay_ms(50); // Wait for more than 40ms after VCC rises to 4.5V
    LCD_SendCommand(0x03);
    _delay_ms(5);  // Wait for more than 4.1ms
    LCD_SendCommand(0x03);
    _delay_us(150);
    LCD_SendCommand(0x03);
    LCD_SendCommand(0x02); // Set to 4-bit mode
    LCD_SendCommand(0x28); // Function set: 4-bit, 2 lines, 5x8 dots
    LCD_SendCommand(0x0C); // Display ON, cursor OFF, blink OFF
    LCD_SendCommand(0x06); // Entry mode set: increment automatically, no shift
    LCD_SendCommand(0x01); // Clear display
    _delay_ms(2);  // Delay after clearing display
}

void LCD_SendCommand(uint8_t cmd) {
    // Send a command to the LCD
    i2c_start(LCD_ADDR<<1);
```

```
    i2c_write((cmd & 0xF0) | 0x08); // Send high nibble
    i2c_write((cmd & 0xF0) | 0x0C); // Enable bit high
    i2c_write((cmd & 0xF0) | 0x08); // Enable bit low
    i2c_write((cmd << 4) | 0x08);   // Send low nibble
    i2c_write((cmd << 4) | 0x0C);   // Enable bit high
    i2c_write((cmd << 4) | 0x08);   // Enable bit low
    i2c_stop();
}

void LCD_SendData(uint8_t data) {
    // Send data to the LCD
    i2c_start(LCD_ADDR<<1);
    i2c_write((data & 0xF0) | 0x09); // Send high nibble
    i2c_write((data & 0xF0) | 0x0D); // Enable bit high
    i2c_write((data & 0xF0) | 0x09); // Enable bit low
    i2c_write((data << 4) | 0x09);   // Send low nibble
    i2c_write((data << 4) | 0x0D);   // Enable bit high
    i2c_write((data << 4) | 0x09);   // Enable bit low
    i2c_stop();
}

void LCD_Print(char *str) {
    while (*str) {
        LCD_SendData(*str++);
    }
}

void rtc_read_time(uint8_t *hours, uint8_t *minutes, uint8_t *seconds) {
    i2c_start(DS3231_ADDRESS<<1);
    i2c_write(0x00); // Start at register 0x00 (seconds)
    i2c_stop();

    i2c_start((DS3231_ADDRESS<<1) | 1);
    *seconds = i2c_read_ack();
    *minutes = i2c_read_ack();
    *hours = i2c_read_nack();
    i2c_stop();
}

void rtc_read_date(uint8_t *day, uint8_t *date, uint8_t *month, uint8_t *year) {
    i2c_start(DS3231_ADDRESS<<1);
    i2c_write(0x03); // Start at register 0x03 (day)
    i2c_stop();

    i2c_start((DS3231_ADDRESS<<1) | 1);
    *day = i2c_read_ack();
    *date = i2c_read_ack();
    *month = i2c_read_ack();
    *year = i2c_read_nack();
    i2c_stop();
}

uint8_t bcd_to_decimal(uint8_t bcd) {
    return ((bcd / 16 * 10) + (bcd % 16));
}

uint8_t decimal_to_bcd(uint8_t decimal) {
    return ((decimal / 10 * 16) + (decimal % 10));
}

void LCD_Printf(const char *fmt, ...) {
    char buffer[32];
    va_list args;
    va_start(args, fmt);
```

```
    vsnprintf(buffer, sizeof(buffer), fmt, args);
    va_end(args);

    char *p = buffer;
    while (*p) {
        LCD_SendData(*p++);
    }
}
```