MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Department of I & CT
MIT, Manipal

RFID-Based Access Control System

ICT 3143 Embedded Systems Lab Mini Project

Vth Sem B.Tech (CCE)

Adriteyo Das (Reg. No - 230953244)
Abhidutta Mukund Giri (Reg. No - 230953232)
Piyush (Reg. No - 230953212)

Under the guidance of

Dr. Santosh Kamath
Associate Professor
Department of I&CT
Manipal Institute of Technology
Manipal, Karnataka, India

November, 2025

ABSTRACT

In an era where security and access control are paramount, RFID (Radio Frequency Identification) technology has emerged as a reliable and efficient solution for authentication and identification systems. This project presents the design and implementation of an RFID-based access control system using the LPC1768 microcontroller, which enables secure registration and verification of RFID cards for controlled access applications. The system demonstrates the practical application of embedded systems in developing modern security solutions that are both cost-effective and user-friendly.

The project design incorporates the LPC1768 microcontroller, chosen for its robust processing capabilities and extensive peripheral support. An MFRC522 RFID reader module serves as the primary input device for reading RFID card UIDs (Unique Identifiers), while a 16x2 LCD display provides real-time feedback to users. A 4x3 matrix keyboard allows users to select between registration and verification modes. An LED indicator provides visual feedback during card scanning operations. Critical methodologies include SPI communication protocol implementation for RFID module interfacing, UID database management in memory, and user interface design for seamless operation.

The system successfully registers RFID card UIDs into an internal database and verifies scanned cards against stored credentials, demonstrating its reliability for access control applications. Results indicate that the project effectively integrates the RFID module, display, and input devices with the microcontroller to create a functional security system. This project's significance lies in its application potential for secure access control in residential, commercial, and institutional settings, where RFID technology provides a convenient and secure alternative to traditional key-based systems. The conclusions drawn from this project emphasize the feasibility and relevance of embedded systems in implementing modern security and authentication protocols.

Contents

List of Figures

1 Introduction

RFID (Radio Frequency Identification) technology has revolutionized the field of contactless identification and access control. This project seeks to implement a practical RFID-based access control system using the LPC1768 microcontroller and MFRC522 RFID reader module. The system provides two primary functionalities: registering new RFID cards into the system's database and verifying scanned cards against registered credentials.

1.1 Brief Description About the Project

The project focuses on implementing an RFID-based access control system using the LPC1768 microcontroller. The system provides two operational modes, which are selected through a matrix keyboard interface:

1. **Registration Mode:**
   - **Input:** RFID card scanned using the MFRC522 reader module. The user selects option 2 from the menu using the matrix keyboard.
   - **Processing:** The system reads the card's UID and checks if it already exists in the database. If not, the UID is stored in memory.
   - **Output:** LCD displays the card UID and confirmation message (”Registered!” or ”Already Exists!” or ”Memory Full!”). LED blinks to confirm successful scan.
2. **Verification Mode:**
   - **Input:** RFID card scanned using the MFRC522 reader module. The user selects option 1 from the menu using the matrix keyboard.
   - **Processing:** The system reads the card's UID and compares it against all stored UIDs in the database.
   - **Output:** LCD displays the card UID and verification result (”Verified!” or ”Not Verified!”). LED blinks to confirm successful scan.

The system can store up to 20 unique RFID card UIDs in its internal memory, making it suitable for small to medium-scale access control applications such as door locks, attendance systems, or secure storage access.

2 Methodology

2.1 Components Required

- **NXP LPC1768 Microcontroller:** The main processor for handling RFID communication, database management, and user interface control. It features ARM Cortex-M3 core, SPI interface support, and sufficient GPIO pins.
- **MFRC522 RFID Reader Module:** Used for reading RFID cards operating at 13.56 MHz. Communicates with the microcontroller via SPI protocol.
- **RFID Cards/Tags:** Passive RFID cards compatible with MFRC522 (ISO 14443A standard) containing unique identification numbers (UIDs).
- **4x3 Matrix Keyboard:** Used for user input to select between registration mode (key 2) and verification mode (key 1).
- **16x2 LCD Display:** Displays system messages, card UIDs, and verification results. Connected in 4-bit mode to minimize GPIO usage.
- **LED Indicator:** Provides visual feedback during card scanning operations, connected to P0.22.
- **Resistors and Capacitors:** Pull-up resistors for keyboard matrix and decoupling capacitors for stable operation.
- **Power Supply:** 5V regulated power supply for the system.

2.2 Block Diagram

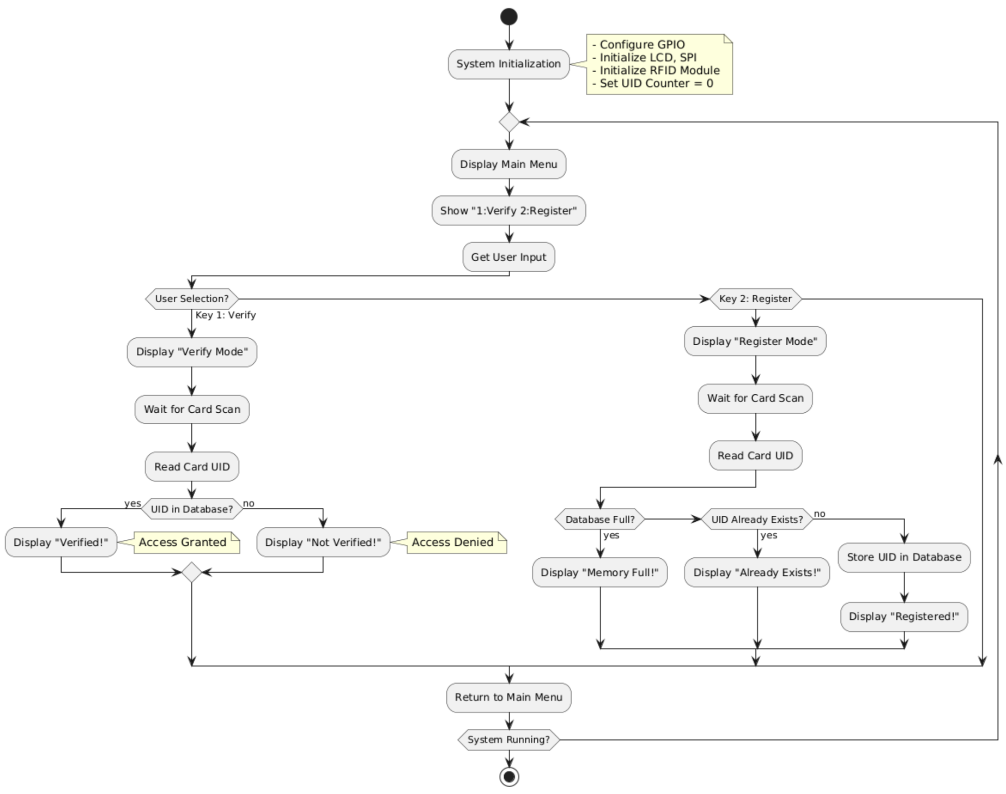

Figure 1: Block diagram of RFID-based Access Control System

The block diagram illustrates the interconnection of all major components. The LPC1768 microcontroller serves as the central processing unit, interfacing with the MFRC522 RFID reader via SPI, the matrix keyboard for input, the LCD for display output, and an LED for visual feedback.

2.3 Connection Description

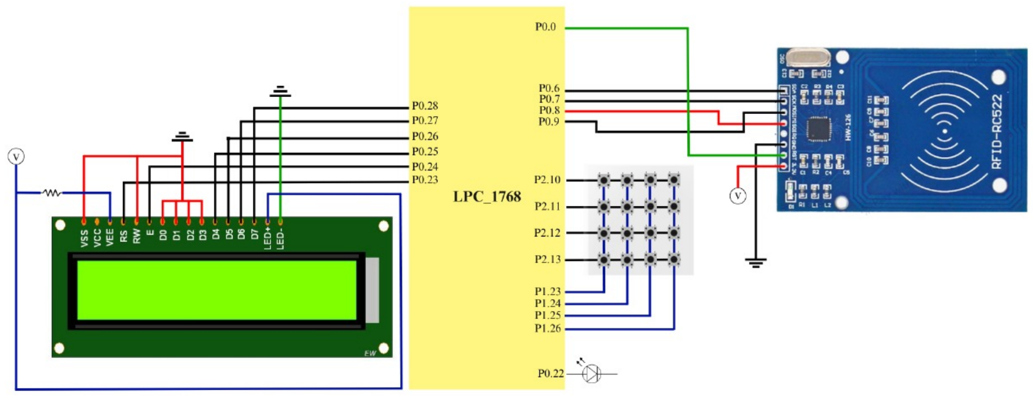

Figure 2: Detailed connection diagram of the system

2.3.1 LCD Connections (4-bit mode)

The LCD is connected to the LPC1768 in 4-bit mode to conserve GPIO pins:

- **Data pins (D4-D7):** Connected to P0.23, P0.24, P0.25, P0.26 respectively
- **RS (Register Select):** Connected to P0.27
- **EN (Enable):** Connected to P0.28
- **RW:** Connected to Ground (write mode only)
- **VSS:** Ground
- **VDD:** +5V
- **V0:** Connected to potentiometer for contrast adjustment

2.3.2 Matrix Keyboard Connections

The 4x3 matrix keyboard is interfaced as follows:

- **Rows (4 pins):** Connected to P2.10, P2.11, P2.12, P2.13 (configured as outputs)
- **Columns (3 pins):** Connected to P1.23, P1.24, P1.25 (configured as inputs with pull-down)

The keyboard scanning is performed by driving rows high one at a time and reading column states.

2.3.3 MFRC522 RFID Module Connections (SPI Interface)

The RFID reader communicates via SPI1 interface:

- **SCK (Serial Clock):** Connected to P0.7 (SSP1 SCK)
- **MISO (Master In Slave Out):** Connected to P0.8 (SSP1 MISO)
- **MOSI (Master Out Slave In):** Connected to P0.9 (SSP1 MOSI)
- **SDA/SS (Slave Select):** Connected to P0.6 (GPIO, active low)
- **RST (Reset):** Connected to +3.3V (always active)
- **VCC:** +3.3V
- **GND:** Ground

2.3.4 LED Indicator

- **LED Anode:** Connected to P0.22 through a current-limiting resistor (220)
- **LED Cathode:** Connected to Ground

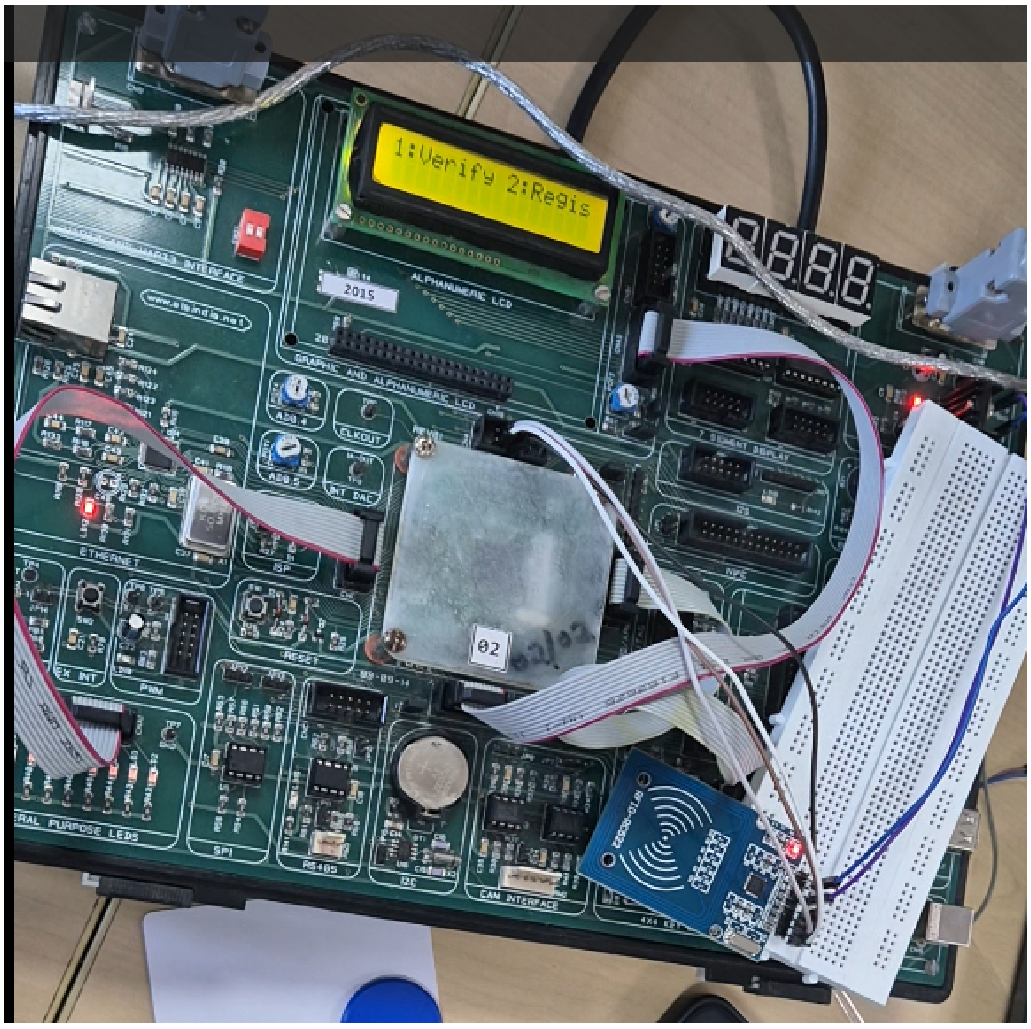

Figure 3: This image shows the fully assembled development board with all sensors and components properly connected and configured. The firmware has been successfully programmed using FlashMagic, ensuring that the system is fully functional for RFID-based access control operations.

2.4 Method

This project designs an RFID-based access control system using the LPC1768 microcontroller with an MFRC522 RFID reader module. The system provides registration and verification modes, allowing users to manage and authenticate RFID cards through a simple menu-driven interface.

2.4.1 System Initialization

Upon power-up, the system performs the following initialization sequence:

1. Configure GPIO pins for LCD (P0.23-P0.28), keyboard matrix (P2.10-P2.13 as outputs, P1.23-P1.25 as inputs), and LED (P0.22).
2. Initialize the LCD display in 4-bit mode and display a welcome message.
3. Configure SPI1 interface for communication with MFRC522 (clock polarity and phase settings, baud rate).
4. Initialize the MFRC522 RFID reader by writing configuration registers for antenna control, timer settings, and operating mode.
5. Set UID database counter to zero (system can store up to 20 UIDs).

2.4.2 Main Menu Interface

The system continuously displays a menu on the LCD: "1:Verify 2:Register"

The matrix keyboard scanning routine polls the first row (keys 1, 2, 3) to detect user selection:

- Key 1: Enter Verification Mode
- Key 2: Enter Registration Mode

2.4.3 Registration Mode

When the user selects key 2, the system enters registration mode:

1. **Display Prompt:** LCD shows "Register Mode" followed by "Scan Card..."
2. **Card Detection:**
   - System continuously sends PICC_REQIDL command (0x26) to detect nearby cards
   - When a card enters the RF field, the MFRC522 responds with card type information
3. **UID Reading:**
   - System sends PICC_ANTICOLL command (0x93) to read the card's unique identifier
   - The 4-byte UID plus 1-byte checksum are received and validated

4. **UID Processing:**
    - Display the UID on LCD in hexadecimal format: "XX XX XX XX XX"
    - Blink LED to confirm successful card read
    - Check if database is full (20 UIDs maximum)
    - Search database to check if UID already exists
    - If UID is new and space is available, store it in the database array
5. **Feedback:**
    - Display "Registered!" if successfully added
    - Display "Already Exists!" if UID was previously registered
    - Display "Memory Full!" if database capacity reached
6. Return to main menu after 4-second delay

2.4.4 Verification Mode

When the user selects key 1, the system enters verification mode:

1. **Display Prompt:** LCD shows "Verify Mode" followed by "Scan Card..."
2. **Card Detection and Reading:** Same process as registration mode to detect card and read UID
3. **UID Verification:**
    - Display the scanned UID on LCD
    - Blink LED to confirm successful card read
    - Compare scanned UID byte-by-byte against all stored UIDs in database
    - Set verification flag if a match is found
4. **Access Decision:**
    - Display "Verified!" if UID exists in database (access granted)
    - Display "Not Verified!" if UID not found in database (access denied)
5. Return to main menu after 4-second delay

2.4.5 SPI Communication Protocol

Communication with the MFRC522 follows this protocol:

- **Write Operation:** Pull SS low, send register address (shifted left, bit 7 = 0), send data byte, pull SS high
- **Read Operation:** Pull SS low, send register address with bit 7 = 1, read data byte, pull SS high
- **Timing:** SPI clock configured at system clock/8 for reliable communication

2.4.6 Database Management

The UID database is implemented as a 2D array: uid_database_rfid[20][5]

- Each row stores one 5-byte UID (4 bytes + 1 checksum)
- Counter variable tracks number of registered cards
- Linear search algorithm used for UID lookup during verification

The system's modular design with separate functions for registration, verification, SPI communication, and LCD display makes it easy to maintain and extend for additional features.

3 Results and Discussions

The RFID-based access control system was successfully implemented and tested with multiple RFID cards. The system demonstrates reliable performance in both registration and verification modes.

3.1 User Interface

Upon startup, the system presents a clear menu allowing users to choose between verification and registration modes using the matrix keypad.

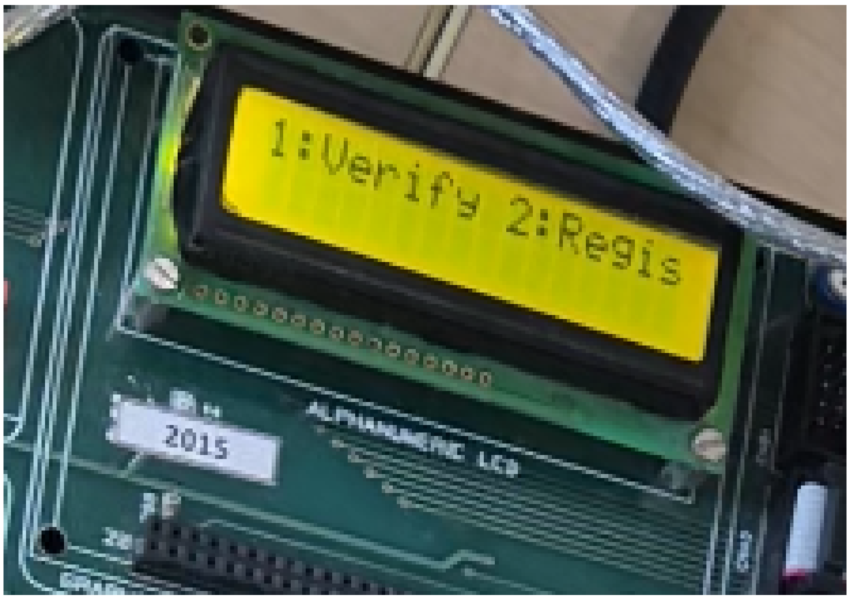

Figure 4: Main menu displayed on LCD prompting user to choose mode

3.2 Registration Mode Results

When a user selects the registration mode (key 2), the system prompts them to scan an RFID card. Once scanned, the UID (5-byte) is read and displayed in hexadecimal format. The system checks whether the UID already exists in the database.

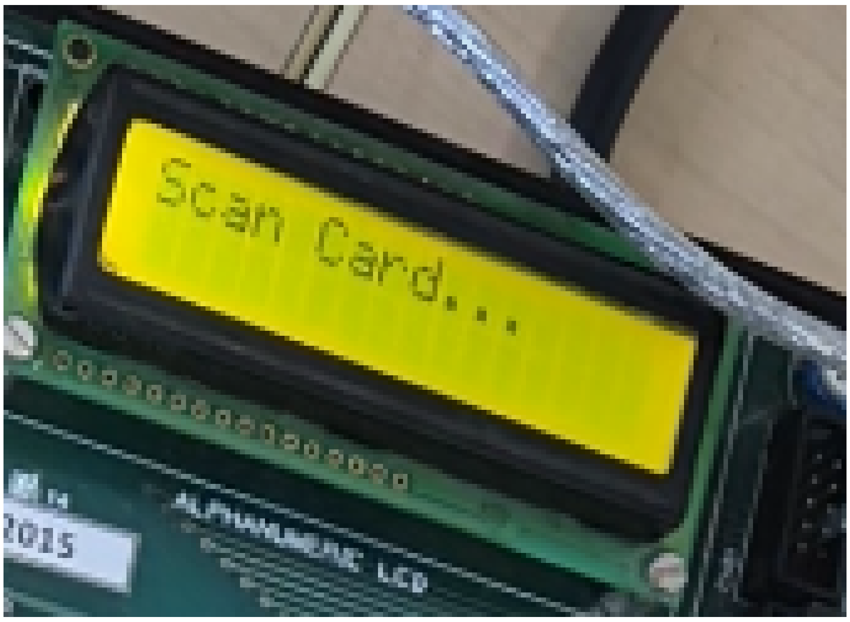

Figure 5: Prompt displayed while scanning card

If the card is new and space is available, the UID is added to the database and a confirmation message is shown.

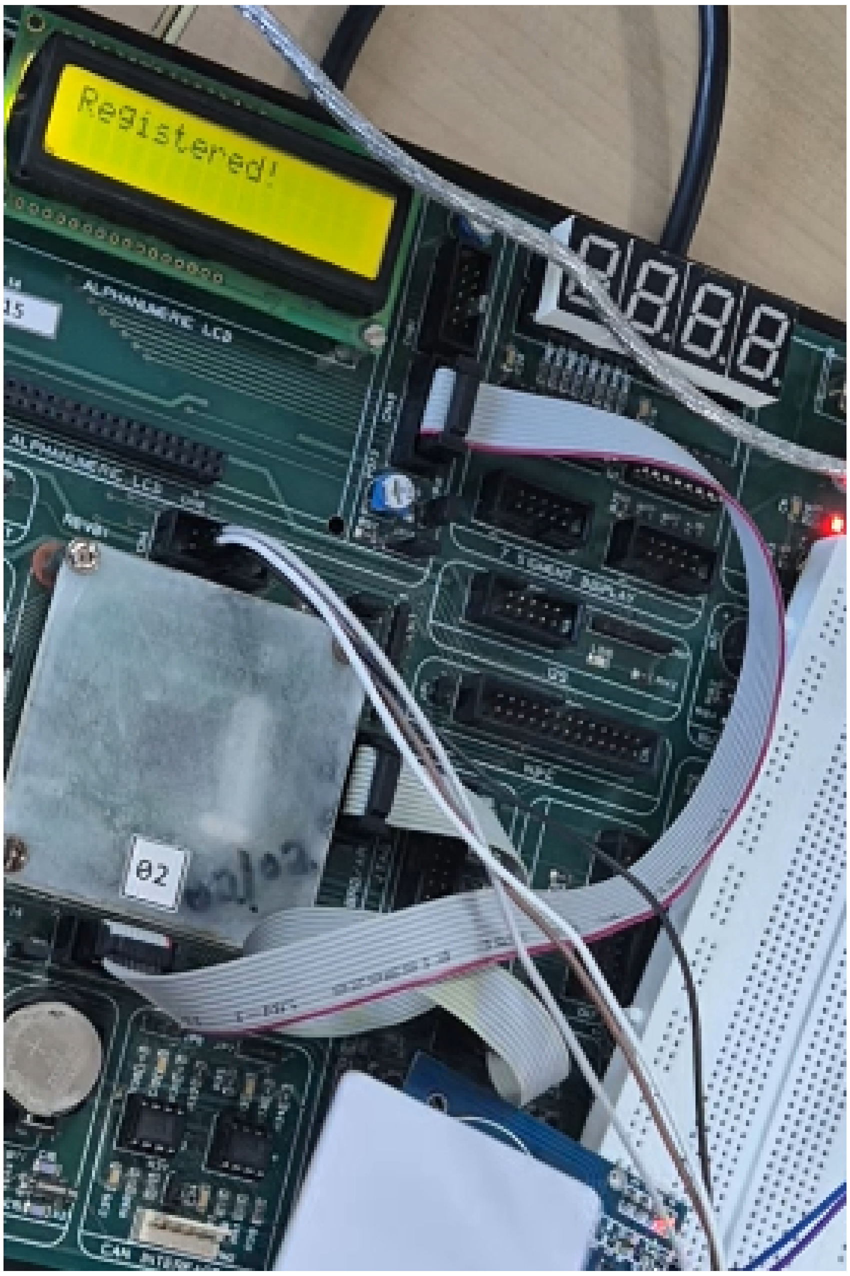

Figure 6: Successful registration confirmation message

If the card is already present in the database, the system prevents duplication and indicates that the card exists.

3.3 Verification Mode Results

When verification mode (key 1) is selected, the user scans a card and the UID is displayed in hexadecimal format. The UID is compared against stored entries in the database.

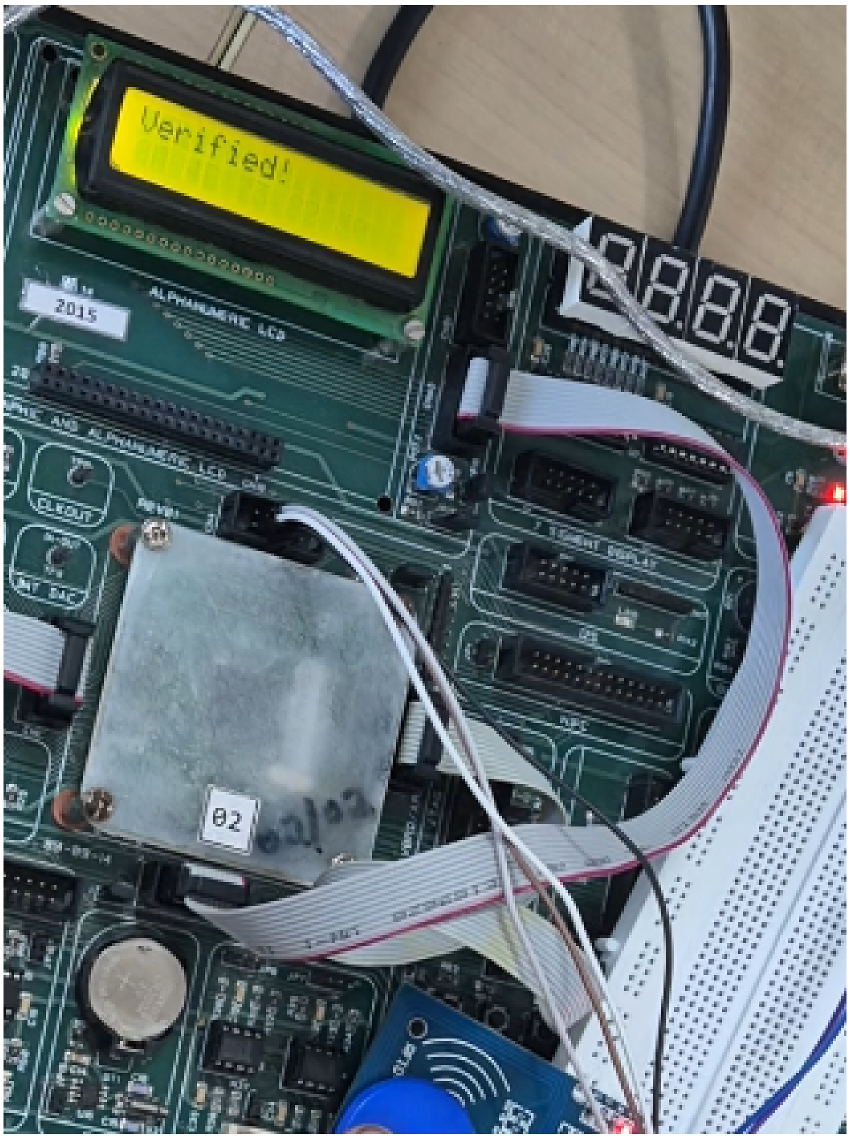

Figure 7: Successful verification - authorized card

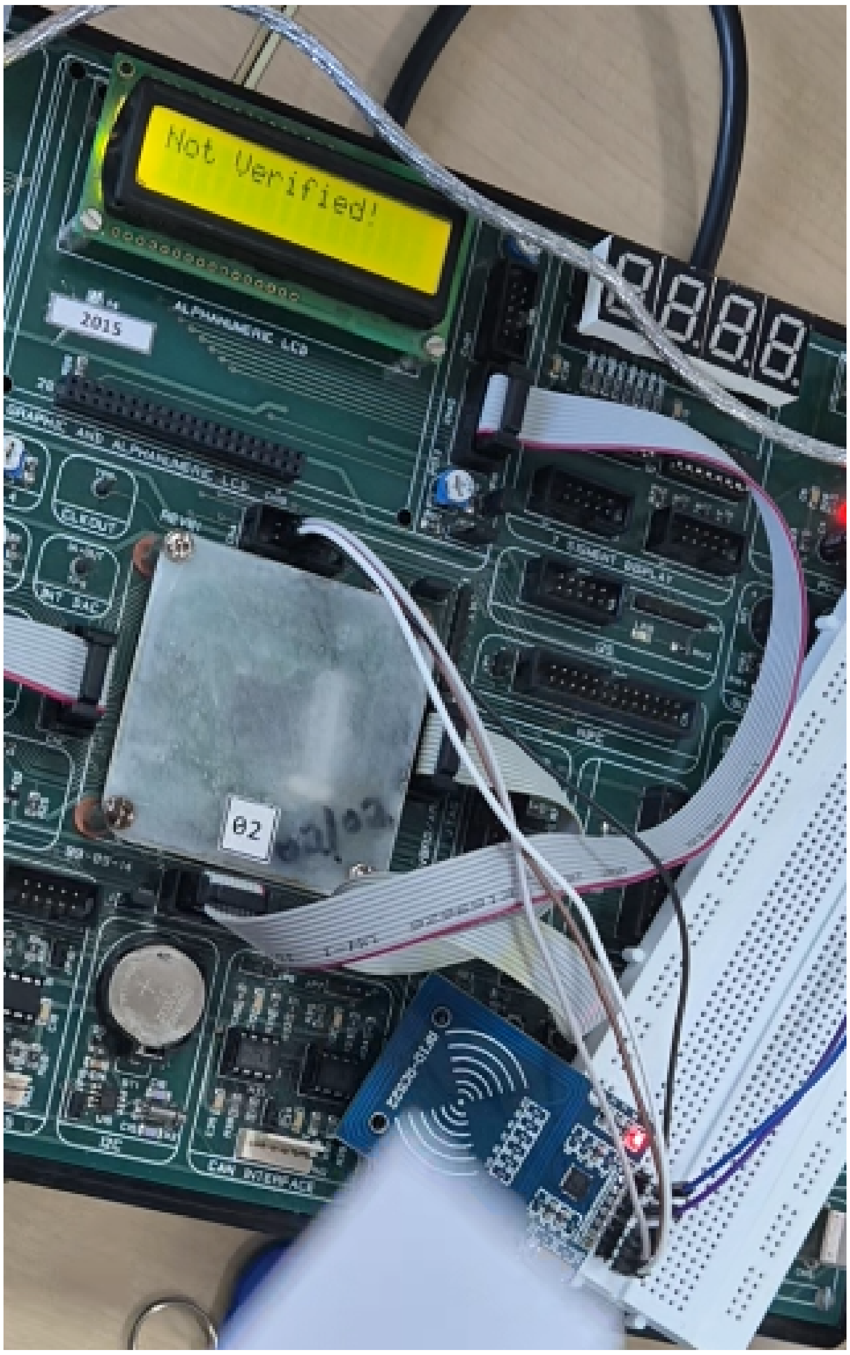

Figure 8: Failed verification - unauthorized card

The system successfully differentiates between registered and unregistered cards and provides immediate LED and LCD feedback.

3.4 System Performance

The system demonstrated the following performance characteristics:

- **Card Detection Time:** Less than 500ms from card placement to UID display
- **Registration Speed:** About 1 second per card including LCD update
- **Verification Speed:** Less than 1 second including search time

- **Database Capacity:** Tested with 20 cards successfully
- **Read Range:** Effective range between 2–5 cm
- **Accuracy:** 100% success in differentiating authorized vs unauthorized cards

3.5 Discussion

The system proves suitable for real-time RFID access control applications using the LPC1768 microcontroller.

Key Observations:

1. Fast and reliable SPI communication with the RFID reader.
2. In-memory UID storage works well for small-scale systems.
3. LCD interface clearly communicates card status and UID in hexadecimal format.
4. Keypad provides simple and intuitive mode selection.
5. LED indicator gives instant confirmation during scanning.

Applications:

- Office/lab security access systems
- Employee/student attendance management
- Smart storage lockers
- Vehicle parking access points
- Library entry and asset tracking

Limitations and Future Enhancements:

- UID storage is volatile; external EEPROM can ensure permanent access records.
- Database limited to 20 cards; can be expanded with larger memory modules.
- No multi-level roles; future work may incorporate admin/user roles.
- No time-based access; RTC can enable schedule-restricted entry.
- No network communication; Wi-Fi/Bluetooth can enable remote monitoring and logging.

3.6 Conclusion

This project successfully demonstrates the implementation of an RFID-based access control system using embedded systems technology. The LPC1768 microcontroller effectively interfaces with the MFRC522 RFID reader module, matrix keyboard, and LCD display to create a functional security system. The project showcases important embedded systems concepts including SPI communication, GPIO interfacing, database management, and real-time user interaction.

The system's reliability, ease of use, and extendibility make it suitable for deployment in various access control scenarios. The modular code structure facilitates future enhancements such as persistent storage, time-based access, and remote monitoring capabilities. This project emphasizes the practical relevance of embedded systems in developing modern security and authentication solutions.

References

1. NXP LPC1768 User Manual, NXP Semiconductors. Available: https://www.keil.com/dd/docs/datashts/philips/lpc17xx_um.pdf

2. MFRC522 Datasheet, NXP Semiconductors. Available: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

3. “Interfacing RFID Sensor-Complete guide for User, ” robu. Available: https://robu.in/interfacing-rfid-sensor-complete-guide/

4. “Understanding SPI Communication Protocol,” SparkFun Electronics. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

5. “RFID-BASED-TOLL-GATE-SYSTEM-USING-LPC1768,” GitHub repository. Available: https://github.com/Bharathg24/RFID-BASED-TOLL-GATE-SYSTEM-USING-LPC1768

4 C Code with Comments

```c
#include "LPC17xx.h"
#include <stdio.h>
#include <string.h>
#include <stdint.h>

/* ----------- LCD / Keypad Globals ----------- */
int temp1, temp2;
int flag_command;
int i, j, r;
signed int row, col;
unsigned int flag;

// Matrix keyboard mapping (4x3)
unsigned int MatrixMap[4][3] = {
    {1, 2, 3},     // Row 0: Keys 1, 2, 3
    {4, 5, 6},     // Row 1: Keys 4, 5, 6
    {7, 8, 9},     // Row 2: Keys 7, 8, 9
    {10, 0, 11}    // Row 3: Keys *, 0, #
};

/* ----------- LED ----------- */
#define LED_PORT LPC_GPIO0
#define LED_PIN  (1 << 22)  // LED connected to P0.22

/* ----------- RFID (MFRC522) Pin Definitions ----------- */
#define SCK  (1 << 7)   // P0.7 - SPI Clock
#define MISO (1 << 8)   // P0.8 - Master In Slave Out
#define MOSI (1 << 9)   // P0.9 - Master Out Slave In
#define SSEL (1 << 6)   // P0.6 - Slave Select (Chip Select)

/* ----------- MFRC522 Register Addresses ----------- */
#define CommandReg         0x01  // Command register
#define ModeReg            0x11  // Mode register
#define TxControlReg       0x14  // TX control register
#define TxAutoReg          0x15  // TX auto register
#define TModeReg           0x2A  // Timer mode register
#define TPrescalerReg      0x2B  // Timer prescaler
#define TReloadRegH        0x2C  // Timer reload high
#define TReloadRegL        0x2D  // Timer reload low
#define CommIEnReg         0x02  // Interrupt enable register
#define BitFramingReg      0x0D  // Bit framing register
#define FIFODataReg        0x09  // FIFO data register
#define FIFOLevelReg       0x0A  // FIFO level register
#define CommIrqReg         0x04  // Interrupt request register
#define ErrorReg           0x06  // Error register
#define ControlReg         0x0C  // Control register

/* ----------- MFRC522 Commands ----------- */
#define PCD_IDLE           0x00  // No action
#define PCD_RESETPHASE     0x0F  // Reset command
#define PCD_TRANSCEIVE     0x0C  // Transceive data

/* ----------- PICC (Card) Commands ----------- */
#define PICC_REQIDL        0x26  // Request Idle cards
#define PICC_ANTICOLL      0x93  // Anti-collision
```

```c

/* ----------- Status Codes ----------- */
#define MI_OK               1
#define MI_ERR              0
#define MI_NOTAGERR         2
#define MAX_LEN             16

/* ----------- UID Database ----------- */
int uid_count_rfid;  // Counter for number of registered cards
unsigned char uid_database_rfid[20][5];  // Storage for up to 20 UIDs
    (5 bytes each)
```

Listing 1: Main Program and Initialization

```c
// Write to LCD port
void port_wr(void)
{
    int jj;
    // Write 4-bit data to P0.23-P0.26
    LPC_GPIO0->FIOPIN = temp2 << 23;

    // Set RS (Register Select) pin
    if (flag_command == 0)
        LPC_GPIO0->FIOCLR = 1 << 27;  // Command mode (RS=0)
    else
        LPC_GPIO0->FIOSET = 1 << 27;  // Data mode (RS=1)

    // Generate enable pulse
    LPC_GPIO0->FIOSET = 1 << 28;  // EN high
    for (jj = 0; jj < 50; jj++);  // Short delay
    LPC_GPIO0->FIOCLR = 1 << 28;  // EN low
    for (jj = 0; jj < 10000; jj++);  // Longer delay
}

// Write 8-bit data as two 4-bit nibbles
void lcd_wr(void)
{
    temp2 = (temp1 >> 4) & 0xF;  // Upper nibble
    port_wr();
    temp2 = temp1 & 0xF;         // Lower nibble
    port_wr();
}

// Simple delay function
void delay(int r1)
{
    for (r = 0; r < r1; r++);
}

// Send command to LCD
void lcd_cmd(unsigned char cmd)
{
    flag_command = 0;  // Command mode
    temp1 = cmd;
    lcd_wr();
    delay(30000);
}

// Send data (character) to LCD
```

```c
void lcd_data(unsigned char data)
{
    flag_command = 1;  // Data mode
    temp1 = data;
    lcd_wr();
    delay(30000);
}

// Display string on LCD
void lcd_string(unsigned char *str)
{
    i = 0;
    flag_command = 1;  // Data mode
    while (str[i] != '\0') {
        temp1 = str[i];
        lcd_wr();
        delay(10000);
        i++;
    }
}

// Clear LCD display
void lcd_clear(void)
{
    lcd_cmd(0x01);  // Clear display command
    delay(50000);
}

// Initialize LCD in 4-bit mode
void lcd_init(void)
{
    int ii;
    // Initialization sequence for 4-bit mode
    int command_init[9] = {3, 3, 3, 2, 2, 0x28, 0x0C, 0x06, 0x01};
    flag_command = 0;
    for (ii = 0; ii < 9; ii++) {
        temp1 = command_init[ii];
        lcd_wr();
        delay(30000);
    }
}
```

Listing 2: LCD Display Functions

```c
// Scan keyboard columns for key press
void scan(void)
{
    unsigned long temp3;
    temp3 = LPC_GPIO1->FIOPIN & 0x03800000;  // Read columns P1.23-P1
        .25
    flag = 0;

    if (temp3 != 0x00000000) {  // If any column is high
        flag = 1;
        // Determine which column
        if (temp3 == (1 << 23)) col = 0;
        else if (temp3 == (1 << 24)) col = 1;
        else if (temp3 == (1 << 25)) col = 2;
    }
}

// Get key press from first row only (keys 1, 2, 3)
int get_first_row_key(void)
{
    int found_key = -1;
    unsigned int col_copy;

    while (1) {
        // Drive first row (P2.10) high
        LPC_GPIO2->FIOPIN = (1 << 10);
        delay(1000);
        scan();

        if (flag == 1) {  // Key pressed
            col_copy = col;
            found_key = MatrixMap[0][col_copy];  // Get key value from
                row 0

            // Wait for key release
            while (1) {
                LPC_GPIO2->FIOPIN = (1 << 10);
                delay(1000);
                scan();
                if (!flag) break;  // Key released
            }
            delay(50000);  // Debounce delay
            return found_key;
        }
    }
}
```

Listing 3: Matrix Keyboard Functions

```c
// Initialize SPI1 for MFRC522 communication
void SPI_INIT(void)
{
    LPC_SC->PCONP |= (1 << 21);  // Power on SSP1

    // Configure pins for SPI1: SCK, MISO, MOSI
    LPC_PINCON->PINSEL0 |= (0x02 << 14) | (0x02 << 16) | (0x02 << 18);

    // Configure SSEL as GPIO output
    LPC_GPIO0->FIODIR |= SSEL;
    LPC_GPIO0->FIOSET = SSEL;  // Initially high (inactive)

    // Configure SPI: 8-bit, SPI mode 0
    LPC_SSP1->CR0 = 0x0707;
    LPC_SSP1->CPSR = 8;  // Clock prescaler
    LPC_SSP1->CR1 = 0x02;  // Enable SSP1
}

// Transfer one byte via SPI
unsigned char SPI_Transfer(unsigned char data)
{
    LPC_SSP1->DR = data;  // Send data
    while (LPC_SSP1->SR & (1 << 4));  // Wait for transfer complete
    return (unsigned char)LPC_SSP1->DR;  // Return received data
}
```

```c

// Chip select control functions
void CS_LOW(void)  { LPC_GPIO0->FIOCLR = SSEL; }
void CS_HIGH(void) { LPC_GPIO0->FIOSET = SSEL; }

// Write to MFRC522 register
void RFID_WriteReg(unsigned char reg, unsigned char val)
{
    CS_LOW();
    SPI_Transfer((reg << 1) & 0x7E);  // Address byte (write)
    SPI_Transfer(val);                 // Data byte
    CS_HIGH();
}

// Read from MFRC522 register
unsigned char RFID_ReadReg(unsigned char reg)
{
    unsigned char val;
    CS_LOW();
    SPI_Transfer(((reg << 1) & 0x7E) | 0x80);  // Address byte (read)
    val = SPI_Transfer(0x00);                   // Read data
    CS_HIGH();
    return val;
}

// Set specific bits in a register
void MFRC522_SetBitMask(unsigned char reg, unsigned char mask)
{
    unsigned char tmp;
    tmp = RFID_ReadReg(reg);
    RFID_WriteReg(reg, tmp | mask);
}

// Clear specific bits in a register
void MFRC522_ClearBitMask(unsigned char reg, unsigned char mask)
{
    unsigned char tmp;
    tmp = RFID_ReadReg(reg);
    RFID_WriteReg(reg, tmp & (~mask));
}

// Turn on RFID antenna
void RFID_AntennaOn(void)
{
    unsigned char val;
    val = RFID_ReadReg(TxControlReg);
    if (!(val & 0x03))
        RFID_WriteReg(TxControlReg, val | 0x03);
}

// Initialize MFRC522
void RFID_Init(void)
{
    RFID_WriteReg(CommandReg, PCD_RESETPHASE);  // Soft reset
    delay(100000);

    // Configure timer
    RFID_WriteReg(TModeReg, 0x8D);
    RFID_WriteReg(TPrescalerReg, 0x3E);
    RFID_WriteReg(TReloadRegL, 0x1E);
    RFID_WriteReg(TReloadRegH, 0);

    // Configure TX
    RFID_WriteReg(TxAutoReg, 0x40);
    RFID_WriteReg(ModeReg, 0x3D);

    RFID_AntennaOn();  // Turn on antenna
}
```

Listing 4: SPI and RFID Communication Functions

```
// Communicate with RFID card
unsigned char RFID_ToCard(unsigned char command, unsigned char *
    sendData,
                          unsigned char sendLen, unsigned char *
                              backData,
                          unsigned int *backLen)
{
    unsigned char status, irqEn, waitIRq, lastBits, n;
    unsigned int i;
    status = MI_ERR;
    irqEn = 0x00;
    waitIRq = 0x00;

    if (command == PCD_TRANSCEIVE) {
        irqEn = 0x77;    // Enable interrupts
        waitIRq = 0x30;  // Wait for RxIRq and IdleIRq
    }

    RFID_WriteReg(CommIEnReg, irqEn | 0x80);
    RFID_WriteReg(CommIrqReg, 0x7F);  // Clear interrupts
    RFID_WriteReg(FIFOLevelReg, 0x80);  // Flush FIFO
    RFID_WriteReg(CommandReg, PCD_IDLE);  // Cancel current command

    // Write data to FIFO
    for (i = 0; i < sendLen; i++)
        RFID_WriteReg(FIFODataReg, sendData[i]);

    // Execute command
    RFID_WriteReg(CommandReg, command);
    if (command == PCD_TRANSCEIVE)
        MFRC522_SetBitMask(BitFramingReg, 0x80);  // Start transmission

    // Wait for command completion
    i = 2000;
    do {
        n = RFID_ReadReg(CommIrqReg);
        i--;
    } while ((i != 0) && !(n & 0x01) && !(n & waitIRq));

    MFRC522_ClearBitMask(BitFramingReg, 0x80);

    if (i != 0) {
        if (!(RFID_ReadReg(ErrorReg) & 0x1B)) {
            status = MI_OK;
            n = RFID_ReadReg(FIFOLevelReg);
            lastBits = RFID_ReadReg(ControlReg) & 0x07;

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

            // Read data from FIFO
            for (i = 0; i < n; i++)
                backData[i] = RFID_ReadReg(FIFODataReg);
            *backLen = n;
        }
    }
    return status;
}

// Request card in field (REQIDL)
unsigned char MFRC522_Request(unsigned char reqMode, unsigned char *
    tagType)
{
    unsigned char status;
    unsigned int backBits;

    RFID_WriteReg(BitFramingReg, 0x07);  // TxLastBits = 7
    tagType[0] = reqMode;
    status = RFID_ToCard(PCD_TRANSCEIVE, tagType, 1, tagType, &backBits
        );
    return status;
}

// Anti-collision detection to get UID
unsigned char MFRC522_Anticoll(unsigned char *serNum)
{
    unsigned char status, i, serNumCheck;
    unsigned int unLen;
    serNumCheck = 0;

    RFID_WriteReg(BitFramingReg, 0x00);  // TxLastBits = 0
    serNum[0] = PICC_ANTICOLL;
    serNum[1] = 0x20;

    status = RFID_ToCard(PCD_TRANSCEIVE, serNum, 2, serNum, &unLen);

    if (status == MI_OK) {
        // Verify checksum (XOR of first 4 bytes should equal 5th byte)
        for (i = 0; i < 4; i++)
            serNumCheck ^= serNum[i];
        if (serNumCheck != serNum[4])
            status = MI_ERR;
    }
    return status;
}
```

Listing 5: RFID Card Communication Functions

```
// Display UID on LCD
void show_uid_on_lcd(unsigned char *serNum)
{
    char buf[20];
    lcd_clear();
    lcd_cmd(0x80);  // First line
    lcd_string("UID:");
```

```c
    lcd_cmd(0xC0);  // Second line
    // Format UID as hex: XX XX XX XX XX
    sprintf(buf, "%02X␣%02X␣%02X␣%02X␣%02X",
            serNum[0], serNum[1], serNum[2], serNum[3], serNum[4]);
    lcd_string((unsigned char*)buf);
    delay(300000);
}

// Search UID in database
int find_uid_rfid(unsigned char *serNum)
{
    int k, m, eq;
    // Check each stored UID
    for (k = 0; k < uid_count_rfid; k++) {
        eq = 1;
        // Compare all 5 bytes
        for (m = 0; m < 5; m++) {
            if (uid_database_rfid[k][m] != serNum[m]) {
                eq = 0;
                break;
            }
        }
        if (eq) return 1;  // UID found
    }
    return 0;  // UID not found
}
```

Listing 6: Helper and Database Functions

```c
// Register new RFID card
void register_uid_rfid(void)
{
    unsigned char status;
    unsigned char serNum[5];
    unsigned char tagType[2];
    int m;

    lcd_clear();
    lcd_string("Register␣Mode");
    delay(200000);
    lcd_clear();
    lcd_string("Scan␣Card...");

    while (1) {
        // Try to detect card
        status = MFRC522_Request(PICC_REQIDL, tagType);

        if (status == MI_OK) {
            // Card detected, read UID
            status = MFRC522_Anticoll(serNum);

            if (status == MI_OK) {
                // Display UID
                show_uid_on_lcd(serNum);

                // Blink LED to confirm scan
                LED_PORT->FIOSET = LED_PIN;
                delay(200000);
                LED_PORT->FIOCLR = LED_PIN;

                // Check if database is full
                if (uid_count_rfid >= 20) {
                    lcd_clear();
                    lcd_string("Memory␣Full!");
                    delay(4000000);
                    return;
                }

                // Check if UID already exists
                if (find_uid_rfid(serNum)) {
                    lcd_clear();
                    lcd_string("Already␣Exists!");
                    delay(4000000);
                    return;
                }

                // Store UID in database
                for (m = 0; m < 5; m++)
                    uid_database_rfid[uid_count_rfid][m] = serNum[m];
                uid_count_rfid++;

                lcd_clear();
                lcd_string("Registered!");
                delay(4000000);
                return;
            }
        }
        delay(50000);  // Wait before next scan
    }
}
```

Listing 7: Registration Mode Function

```
// Verify RFID card against database
void verify_uid_rfid(void)
{
    unsigned char status;
    unsigned char serNum[5];
    unsigned char tagType[2];

    lcd_clear();
    lcd_string("Verify␣Mode");
    delay(200000);
    lcd_clear();
    lcd_string("Scan␣Card...");

    while (1) {
        // Try to detect card
        status = MFRC522_Request(PICC_REQIDL, tagType);

        if (status == MI_OK) {
            // Card detected, read UID
            status = MFRC522_Anticoll(serNum);

            if (status == MI_OK) {
                // Display UID
                show_uid_on_lcd(serNum);

                // Blink LED to confirm scan
                LED_PORT->FIOSET = LED_PIN;
                delay(200000);
                LED_PORT->FIOCLR = LED_PIN;

                // Check if UID exists in database
                lcd_clear();
                if (find_uid_rfid(serNum))
                    lcd_string("Verified!");     // Access granted
                else
                    lcd_string("Not␣Verified!"); // Access denied

                delay(4000000);
                return;
            }
        }
        delay(50000);  // Wait before next scan
    }
}
```

Listing 8: Verification Mode Function

```
// Main program
int main(void)
{
    int key;
    unsigned char msg1[] = "1:Verify␣2:Register";

    SystemInit();
    uid_count_rfid = 0;  // Initialize UID counter

    // Configure GPIO pins
    LPC_GPIO0->FIODIR |= 0xF << 23 | 1 << 27 | 1 << 28;  // LCD pins
    LPC_GPIO2->FIODIR = 0x00003C00;  // Keyboard rows (output)
    LPC_GPIO1->FIODIR = 0;           // Keyboard columns (input)
    LED_PORT->FIODIR |= LED_PIN;     // LED pin (output)
    LED_PORT->FIOCLR = LED_PIN;      // Turn off LED initially

    // Initialize peripherals
    lcd_init();
    lcd_clear();
    SPI_INIT();
    RFID_Init();

    // Main loop
    while (1) {
        // Display menu
        lcd_clear();
        lcd_string(msg1);

        // Wait for key press (1 or 2)
        key = get_first_row_key();

        if (key == 1) {
            verify_uid_rfid();    // Verification mode
        } else if (key == 2) {
            register_uid_rfid();  // Registration mode
        }
    }
}
```

Listing 9: Main Function