Smart Digital Door Lock System

Secure Access Control using PIC Microcontroller

Project Overview

Objective:

- Design a secure, keyless entry system.
- Implement distinct User and Admin access levels.

Key Features:

- 4-Digit PIN Authentication.
- Persistent Data Storage (EEPROM).
- Security Lockout Mechanism.

Hardware Architecture

Core Components:

Microcontroller

PIC16F877A (The "Brain").

Input

Keypad.

Output

16x2 LCD Display.

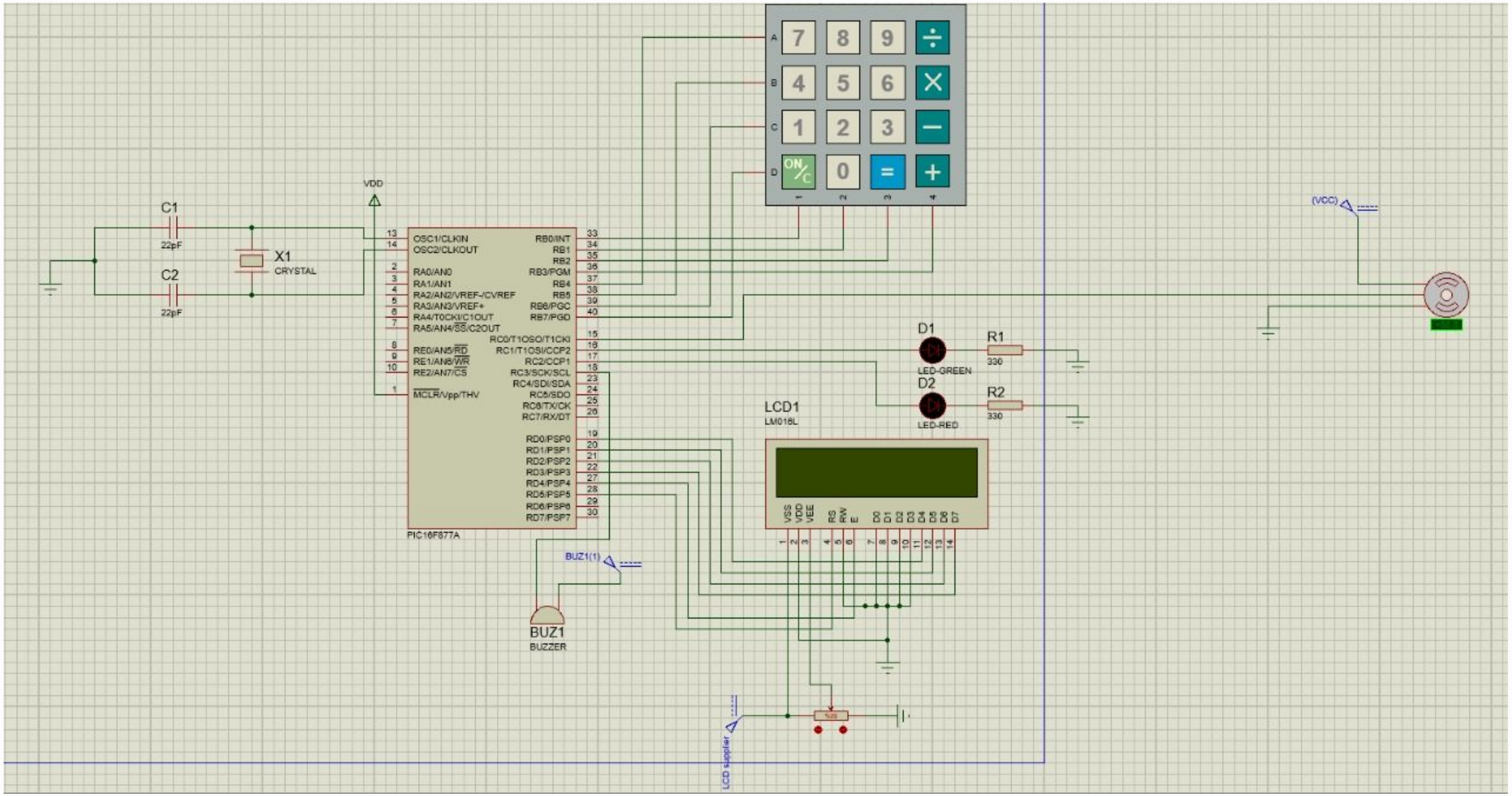
Actuator

Servo Motor.

Storage

Internal EEPROM.

Circuit Design (Proteus)



Project Live Demonstration

Code Architecture – Control Flow (Part 1)

1. System Entry Point

void main(): The central scheduler. It initializes hardware (LCD, Keypad, Timer) and enters an infinite loop to switch between system states.

2. State Handler Functions

void login_mode()

Default state. Manages user authentication and grants access upon success.

void admin_mode()

Secured menu. Verifies the master key ("0000") before allowing password changes.

void suspended_mode()

Penalty state. Traps the system in a hardware timer loop when security is breached.

Flow Diagram:

main() → Switch(State) → login_mode() / admin_mode() / suspended_mode()

Code Architecture – Operational Logic (Part 2)

3. Action Functions

void add_user(): Search algorithm that finds the first 0xFF (empty) slot in EEPROM to store new credentials.

void delete_user(): Memory management function that targets a specific ID and resets its slot to 0xFF.

int read_password(): Input handler that captures 4 keys, handles masking (*), and manages the "Back" button.

4. Hardware Drivers

void Servo_Move(int angle): Custom signal generator creating PWM pulses to control the lock.

void Lcd_Out_Const(...): Memory optimizer fetching text from ROM (Flash) to RAM for display.

System Logic (State Machine)

The software relies on a state machine architecture to manage system modes efficiently and prevent invalid operations.

STATE_LOGIN

Default user access

STATE_ADMIN

Management menu

STATE_LOCKED

Penalty mode

Code Snippet:

```
enum SystemState {
 STATE_LOGIN, // Default user access
 STATE_ADMIN, // Management menu
 STATE_LOCKED // Penalty mode
};

enum SystemState current_state = STATE_LOGIN;
```

System Logic (State Machine)

Code Snippet:

```
while (1)
   {
       switch (current_state)
       {
       case STATE_LOGIN:
           login_mode();
           break;

       case STATE_ADMIN:
           admin_mode();
           break;

       case STATE_LOCKED:
           suspended_mode();
           break;
       }
   }
```

```
enum SystemState toggleState = 1 - current_state; // toggling state
```

User Mode Initialisation

Default Behaviour:

Startup

System defaults to STATE_LOGIN.

LED Status

Red LED (RC2) is set ON to indicate the door is locked; Green LEDs are OFF.

One-Time Display

The code uses an if (!user) check to display the "USER MODE" banner only once, preventing screen flickering.

Code Snippet:

```
// Initialize LEDs (Red ON, Green OFF)
portC.f0 = 0;
portC.f1 = 0;
portC.f2 = 1;

if (!user) {
 Lcd_Out_Const(1, 1, msg_user_mode);
 user = 1; // Set flag to avoid re-printing
}
```

User Mode (Authentication Logic)

The system scans the EEPROM memory to verify the entered password against all saved users.

Code Snippet:

```
// Scan all 64 memory slots (256 byte / 4 bytes per slot = 64 user)
for (slot = 0; slot < 64; slot++) {
    addr = slot * 4;
    // ... Read stored_pass from EEPROM ...

    if (strcmp(password_input, stored_pass) == 0) {
        match_found = 1;  // Access Granted
        break;
    }
}
```

Technical Highlight (Software PWM)

To control the servo motor without using a dedicated PWM pin, we implemented a custom signal generator function.

Code Snippet:

```
void Servo_Move(int angle) {
    // Calculate pulse width based on angle
    on_time = 600 + (angle * 10);

    Servo_Pin = 1;
    VDelay_us(on_time);  // High Pulse

    Servo_Pin = 0;
    Delay_ms(17);        // Low for remainder of 20ms cycle
}
```

Security (Suspended Mode)

If the system is locked, a dedicated hardware timer loop enforces a **60-second delay**, preventing brute-force attacks.

Code Snippet:

```
while (seconds > 0) {
 // Wait for Timer0 Interrupt Flag
 if (INTCON.TMR0IF == 1) {
 INTCON.TMR0IF = 0; // Reset flag
 i++;
 }
 // Update LCD countdown...
}
```

Security (Tries Limit)

Brute Force Protection:

Limit: The system allows exactly **3 attempts**.

Counter: tries_left initialises at 3.

Trigger: On every wrong entry, the counter decrements. When it hits 0, the state changes to STATE_LOCKED.

Code Snippet:

```
tries_left--;
if (tries_left <= 0) {
  current_state = STATE_LOCKED;  // Enforce Penalty
  return;
}
```

Admin Mode (Add User)

Admins can add users. The system automatically finds the first available "empty" slot (indicated by 0xFF) to maximise memory usage.

Code Snippet:

```
// Find first empty slot
for (slot = 0; slot < 64; slot++) {
 addr = slot * 4;

 if (EEPROM_Read(addr) == 0xFF) {
 is_full = 0;
 break; // Found space for new user
 }
}
```

Admin Mode (Delete User)

Deletion Logic:

To "delete" a user, we simply overwrite their memory slot with 0xFF rather than shifting the entire database.

Why 0xFF?

EEPROM Standard

An unprogrammed EEPROM byte is 11111111 (Binary) or 0xFF (Hex).

Efficiency

Resetting a slot to 0xFF marks it as "Empty," allowing the add_user function to reuse it later.

Code Snippet:

```
addr = id * 4;  // Calculate address

EEPROM_Write(addr, 0xff);  // Reset slot to "Empty"

Lcd_Out_Const(1, 1, msg_deleted);
```

Challenges & Solutions

Problem: RAM Scarcity

The PIC16F877A microcontroller possesses extremely limited RAM. Storing various display messages like "Welcome" or "Wrong Password" as standard variables would quickly exhaust available memory, leading to system instability.

This constraint is critical in embedded systems where resources are tightly managed.

Solution: ROM Storage

All display strings were declared as `const`, compelling the compiler to store them in the more abundant Flash Memory (ROM).

A custom `Lcd_Out_Const()` helper function was implemented. This function temporarily copies the required string from ROM to a small RAM buffer for display, preventing continuous RAM occupation.

Code Snippet:

```
// Strings stored in ROM to save RAM
const char msg_welcome[] = "WELCOME";
const char msg_wrong_pin[] = "WRONG PIN";

// Helper function to print from ROM to LCD
void Lcd_Out_Const(char row, char col, const char *text) {
 // Copies 'text' from ROM to a small RAM buffer, then displays it
}
```

Challenges & Solutions

Challenges & Solutions:

Problem Ram Scarcity

Why 0xFF?

EEPROM Standard

An unprogrammed EEPROM byte is 11111111 (Binary) or 0xFF (Hex).

Efficiency

Resetting a slot to 0xFF marks it as "Empty," allowing the add_user function to reuse it later.

Code Snippet:

```
addr = id * 4;  // Calculate address

EEPROM_Write(addr, 0xff);  // Reset slot to "Empty"

Lcd_Out_Const(1, 1, msg_deleted);
```

Conclusion

Summary:

The project successfully demonstrates embedded systems concepts: **State Machines**, **Non-Volatile Memory**, and **Signal Processing**.

Future Improvements:

RFID Integration: For faster access.

Battery Backup: To ensure operation during power cuts.

Thank You

We appreciate your attention and hope our Smart Digital Door Lock System presentation provided valuable insights.

The Team:

- Amira Salah Mohamed Selim Ramon
- Raghad Tariq Amin Ali Abdullah Hassouna
- Rodina Hossam Ahmed Abbas Hamza
- Abdullah Ahmed Mohamed Al-Saeed Al-Shishtawy
- Amr Khaled Mostafa El-Hefnawy
- Mostafa Ahmed Mostafa Khaira
- Mohannad Mohamed Al-Sayed Abdelkarim
- Youssef Mohamed Atallah Hegazy