Faculty of Computer Applications & Information Technology

iMCA PROGRAMME

Semester III

Project Report for

242301306 Introduction to Embedded Systems

Project Definition

Bank Security System with Time-Access Control

Submitted by:

Group No:
Jahanvi S Adhav (202402519010231)

1. ABSTRACT

Purpose: This project aims to develop an integrated security system that provides dual-layer protection by combining real-time perimeter monitoring with a secure, time-based access control mechanism. Its primary purpose is to create a cost-effective solution for safeguarding sensitive areas against unauthorized intrusion and access.

Methodology: The system is built around an Arduino Uno microcontroller. For perimeter security, an HC-SR04 ultrasonic sensor continuously measures distance, triggering a multi-level alarm (using an LED and buzzer) when objects breach predefined zones. For access control, a DS3231 RTC module generates a dynamic password that changes every minute. User authentication is handled via a 4x4 matrix keypad, and all system status information is displayed on a 20x4 I2C LCD. A solenoid lock is activated upon successful password verification.

Expected Outcome: The successful integration of these components is expected to result in a robust and reliable automated system. The final outcome is a fully functional device that effectively monitors a perimeter for intrusions while providing a highly secure, dynamic password-based entry system, demonstrating a practical and enhanced security solution.

2. OBJECTIVE

The main aim of this project is to develop an integrated security system that solves the limitations of conventional single-function security devices. It addresses the problem of passive monitoring by combining real-time perimeter intrusion detection with an active, time-based access control mechanism. The motivation behind developing this system is to create a cost-effective solution that provides comprehensive security for sensitive areas. By merging dual security layers into one automated unit, the project enhances protection against unauthorized access while maintaining user convenience through dynamic password authentication.

3. COMPONENT LIST WITH FUNCTIONALITY

Hardware Components:-

Arduino Uno – Microcontroller board used as the main brain for controlling all project components and processing logic.

HC-SR04 Ultrasonic Sensor – Used for measuring distance to objects and detecting intrusions in the perimeter security system.

DS3231 RTC Module – Provides accurate real-time colock data for generating time-based dynamic passwords.

20x4 I2C LCD Display – Displays system status, messages, distance measurements, time, and password information to the user.

4x4 Matrix Keypad – Allows users to input passwords and interact with the access control system.

5V Solenoid Lock – Acts as the door lock actuator that engages/disengages based on successful authentication.

Buzzer – Provides audible alerts and feedback for different security zones and system events.

LED – Serves as a visual indicator for warning and alert status during security monitoring.

Power Supply – Provides the required 5V DC voltage and current to power the entire circuit.

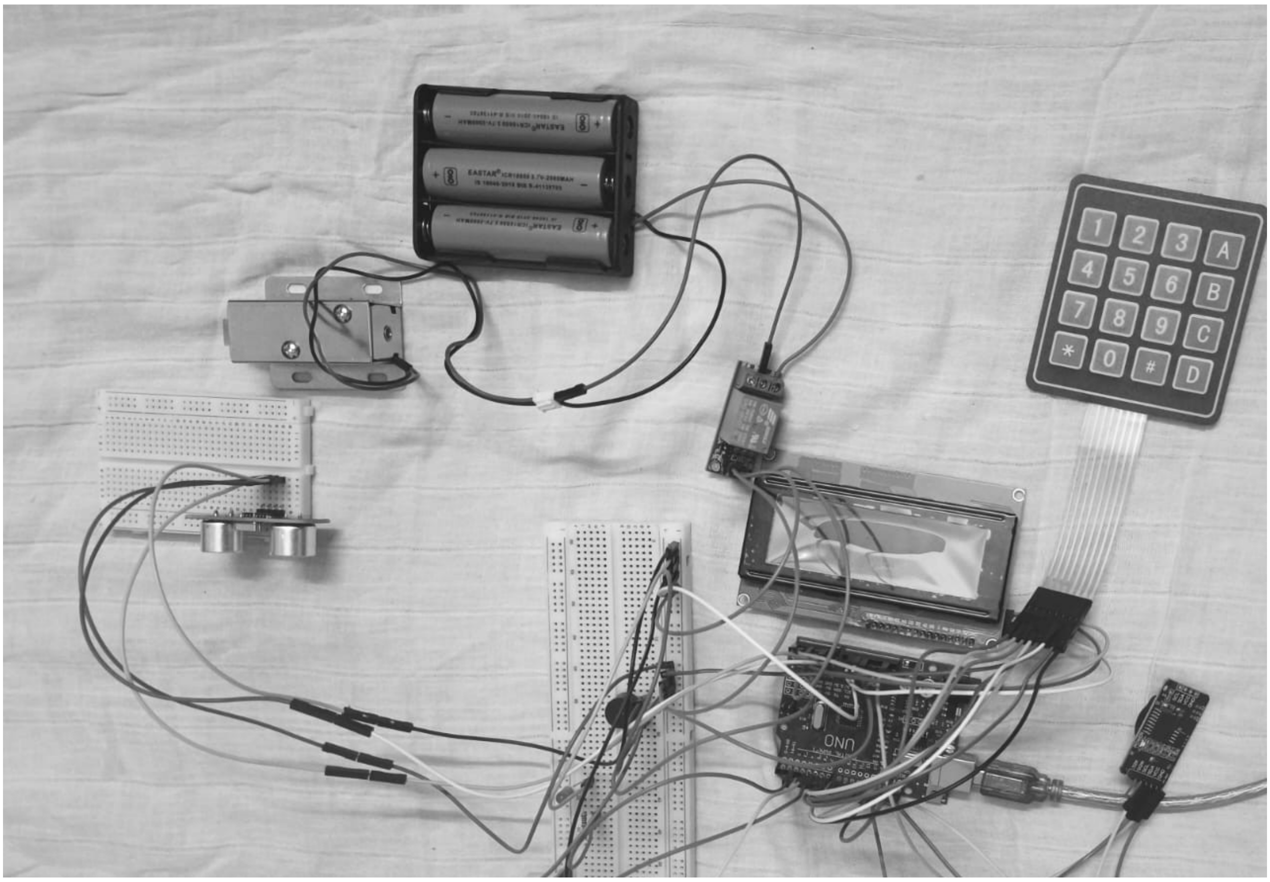
Jumper Wires & Breadboard – Used for making electrical connections between components during prototyping.

Software Components:-

Arduino IDE:- Used for writing, compiling , and Uploading the program code to The Arduino UNO micro- Controller.

Embedded C++:- Programming language used for the Implementation for project

4. Connection Diagram



5. PROJECT CODE SNIPET WITH COMMENTS TO EXPLAIN CODE SECTIONS

```
#include <Wire.h>
#include <Keypad.h>
#include <RTClib.h>
#include <LiquidCrystal_I2C.h>

// ========== ULTRASONIC SENSOR SETUP ==========
#define trigPin1 5      // First ultrasonic sensor
#define echoPin1 4
#define trigPin2 A1     // Second ultrasonic sensor
#define echoPin2 A0
#define buzzerPin A3    // Buzzer pin
#define redLedPin 2     // Red LED pin

// Alert thresholds - SENSOR 1 THRESHOLD SET TO 30cm
const int SENSOR1_ALERT_DISTANCE = 30;   // cm - Sensor 1 specific distance
const int HIGH_ALERT_DISTANCE = 20;      // cm - High alert distance for sensor 2
const int WARNING_DISTANCE = 50;         // cm - Warning distance

// System state variables
bool systemOn = true;  // System always on

// Ultrasonic sensor data
long distance1 = 0;
long distance2 = 0;

// ========== LCD I2C SETUP ==========
LiquidCrystal_I2C lcd(0x27, 20, 4);  // Address 0x27, 20 columns, 4 rows

// ========== KEYPAD SETUP ==========
const byte ROWS = 4; // Four rows
const byte COLS = 4; // Four columns

char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {13, 12, 11, 10}; // Connect to the row pinouts of the keypad
byte colPins[COLS] = {9, 8, 7, 6};   // Connect to the column pinouts of the keypad

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

// ========== RTC SETUP ==========
RTC_DS3231 rtc;

// ========== ACCESS CONTROL SETUP ==========
// Normal code settings
String normalCode = "1234"; // Base normal code (4 digits)
String inputPassword = "";
```

```
// Solenoid lock pin
const int SOLENOID_PIN = A2;

// Access control
bool accessGranted = false;
bool systemLocked = false;
unsigned long accessTime = 0;
unsigned long lockTime = 0;
const unsigned long ACCESS_DURATION = 10000; // 10 seconds access
const unsigned long LOCKOUT_DURATION = 10000; // 10 seconds lockout

// Attempt counter
int attemptCount = 0;
const int MAX_ATTEMPTS = 3;

// For countdown display
unsigned long lastDisplayTime = 0;
unsigned long lastLCDUpdate = 0;
unsigned long lastSensorRead = 0;
const unsigned long LCD_UPDATE_INTERVAL = 500; // Update LCD every 500ms
const unsigned long SENSOR_READ_INTERVAL = 200; // Read sensors every 200ms

// LCD display states
enum LCDState {
  STATE_IDLE,
  STATE_INPUT,
  STATE_ACCESS_GRANTED,
  STATE_ACCESS_DENIED,
  STATE_SYSTEM_LOCKED,
  STATE_ULTRASONIC_ALERT
};
LCDState currentLCDState = STATE_IDLE;
LCDState previousLCDState = STATE_IDLE;

void setup() {
  Serial.begin(9600);

  // Initialize ultrasonic sensor pins
  pinMode(trigPin1, OUTPUT);
  pinMode(echoPin1, INPUT);
  pinMode(trigPin2, OUTPUT);
  pinMode(echoPin2, INPUT);

  // Initialize buzzer
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(buzzerPin, LOW);

  // Initialize red LED
  pinMode(redLedPin, OUTPUT);
  digitalWrite(redLedPin, LOW);

  // Initialize solenoid lock pin
  pinMode(SOLENOID_PIN, OUTPUT);
  digitalWrite(SOLENOID_PIN, HIGH); // Ensure lock is initially closed

  // Initialize I2C bus
  Wire.begin();

  // Initialize LCD - 20x4 display
  lcd.begin(20, 4);
  lcd.backlight();
  lcd.clear();

  // Display startup message across 4 lines
  lcd.setCursor(2, 0);
  lcd.print(F("Bank Security"));
  lcd.setCursor(4, 1);
  lcd.print(F("System"));
  lcd.setCursor(2, 2);
  lcd.print(F("=================="));
  lcd.setCursor(4, 3);
  lcd.print(F("Starting..."));

  // Initialize RTC
  if (!rtc.begin()) {
    Serial.println(F("Couldn't find RTC"));
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("RTC Error!"));
    lcd.setCursor(0, 1);
    lcd.print(F("Check RTC Module"));
    while (1);
  }

  // Set RTC time if it's not running
  if (rtc.lostPower()) {
    Serial.println(F("RTC lost power, setting time"));
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }

  // Play startup sound
  playStartupTone();

  delay(2000);

  // Show idle screen
  updateIdleScreen();

  Serial.println(F("Bank Security System with Dual Sensors Initialized"));
}

void loop() {
  // ========== ULTRASONIC SENSOR LOGIC ==========
  // Read sensors at regular intervals to avoid interference
  if (millis() - lastSensorRead > SENSOR_READ_INTERVAL) {
    lastSensorRead = millis();
    // Read both ultrasonic sensors
    distance1 = getDistance(trigPin1, echoPin1);
    distance2 = getDistance(trigPin2, echoPin2);

    // Print sensor readings to Serial
    Serial.print(F("Sensor 1: "));
    Serial.print(distance1);
    Serial.print(F(" cm | Sensor 2: "));
    Serial.print(distance2);
    Serial.println(F(" cm"));
  }

  // Control buzzer and LED based on sensor readings
  controlBuzzerAndLed(distance1, distance2);

  // Check if we need to show ultrasonic alert on LCD
  if ((distance1 <= SENSOR1_ALERT_DISTANCE || distance2 <= HIGH_ALERT_DISTANCE) &&
      currentLCDState != STATE_ACCESS_GRANTED) {
    currentLCDState = STATE_ULTRASONIC_ALERT;
  } else if (distance1 > SENSOR1_ALERT_DISTANCE && distance2 > HIGH_ALERT_DISTANCE
&&
             currentLCDState == STATE_ULTRASONIC_ALERT) {
    currentLCDState = STATE_IDLE;
  }

  // ========== KEYPAD ACCESS LOGIC ==========
  char key = keypad.getKey();

  if (key && !systemLocked) {
    // Play key press sound
    playKeyTone();

    if (key == '*') {
      inputPassword = ""; // Clear input
      Serial.println(F("Input cleared"));
      updateInputScreen();
    }
    else if (key == '#') {
      checkPassword();
    }
    else if (key >= '0' && key <= '9') {
      // Only accept numbers for password input
      if (inputPassword.length() < 4) {
        inputPassword += key;
        Serial.print(F("Current input: "));
        for (int i = 0; i < inputPassword.length(); i++) {
          Serial.print('*');
        }
        Serial.println();
        updateInputScreen();
      } else {
        Serial.println(F("Maximum password length reached (4 digits)"));
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(F("Max Length: 4 Digits"));
        lcd.setCursor(0, 1);
        lcd.print(F("Press # to submit"));
        lcd.setCursor(0, 2);
        lcd.print(F("or * to clear"));
        playAccessDeniedTone();
      }
    }
  }
  else if (key && systemLocked) {
    // System is locked
    Serial.println(F("System locked! Please wait..."));
    playAccessDeniedTone();
  }

  // Check if access time has expired
  if (accessGranted && (millis() - accessTime > ACCESS_DURATION)) {
    accessGranted = false;
    digitalWrite(SOLENOID_PIN, HIGH); // Lock the solenoid
    Serial.println(F("Access expired - Lock engaged"));
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("Access Expired"));
    lcd.setCursor(0, 1);
    lcd.print(F("Door Locked"));
    playAccessExpiredTone();
    delay(2000);
    currentLCDState = STATE_IDLE;
  }

  // Check if lockout time has expired
  if (systemLocked && (millis() - lockTime > LOCKOUT_DURATION)) {
    systemLocked = false;
    attemptCount = 0;
    Serial.println(F("System unlocked. You can try again."));
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("System Unlocked"));
    lcd.setCursor(0, 1);
    lcd.print(F("Try Again"));
    playSystemUnlockedTone();
    delay(2000);
    currentLCDState = STATE_IDLE;
  }

  // Display countdowns every second
  if (millis() - lastDisplayTime > 1000) {
    lastDisplayTime = millis();

    // Display access countdown if access is granted
    if (accessGranted) {
      displayAccessCountdown();
    }

    // Display lockout countdown if system is locked
    if (systemLocked) {
      displayLockoutCountdown();
    }
  }

  // Update LCD display at regular intervals
  if (millis() - lastLCDUpdate > LCD_UPDATE_INTERVAL) {
    lastLCDUpdate = millis();
    updateLCDDisplay();
  }

  delay(100);
}

// ========== ULTRASONIC SENSOR FUNCTIONS ==========
long getDistance(int trig, int echo) {
  long duration, distance;

  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  duration = pulseIn(echo, HIGH);
  distance = duration * 0.034 / 2;

  // Filter out erroneous readings
  if (distance > 400 || distance < 0) {
    return 999; // Return a large value for out-of-range readings
  }

  return distance;
}

void controlBuzzerAndLed(long dist1, long dist2) {
  // Check for Sensor 1 specific alert (30cm)
  if (dist1 <= SENSOR1_ALERT_DISTANCE) {
    // Sensor 1 alert - buzzer and red LED
    tone(buzzerPin, 1000);
    digitalWrite(redLedPin, HIGH);
    Serial.println(F("SENSOR 1 ALERT! Object at 30cm!"));
  }
  // Check for high alert from sensor 2
  else if (dist2 <= HIGH_ALERT_DISTANCE) {
    // Sensor 2 high alert - red LED only (blinking), no buzzer
    noTone(buzzerPin); // Ensure buzzer is off

    // Blink red LED
    if (millis() % 500 < 250) {
      digitalWrite(redLedPin, HIGH);
    } else {
      digitalWrite(redLedPin, LOW);
    }
    Serial.println(F("SENSOR 2 ALERT! Red LED blinking"));
  }
  // Check for warning from sensor 1
  else if (dist1 <= WARNING_DISTANCE) {
    // Sensor 1 warning - intermittent buzzer and red LED
    if (millis() % 500 < 250) {
      tone(buzzerPin, 800);
      digitalWrite(redLedPin, HIGH);
    } else {
      noTone(buzzerPin);
      digitalWrite(redLedPin, LOW);
    }
    Serial.println(F("Warning: Object approaching Sensor 1"));
  }
  // Check for warning from sensor 2
  else if (dist2 <= WARNING_DISTANCE) {
    // Sensor 2 warning - red LED only (solid), no buzzer
    noTone(buzzerPin);
    digitalWrite(redLedPin, HIGH); // Solid red LED
    Serial.println(F("Warning: Object approaching Sensor 2"));
  }
  else {
    // Safe distance - no sound, LED off
    noTone(buzzerPin);
    digitalWrite(redLedPin, LOW);
    Serial.println(F("Safe distance - No objects detected"));
  }
}

// ========== ACCESS CONTROL FUNCTIONS ==========
void updateLCDDisplay() {
  // Only update if state changed or for time-based updates
  if (currentLCDState != previousLCDState) {
    previousLCDState = currentLCDState;

    switch (currentLCDState) {
      case STATE_IDLE:
        updateIdleScreen();
        break;
      case STATE_INPUT:
        updateInputScreen();
        break;
      case STATE_ACCESS_GRANTED:
        updateAccessGrantedScreen();
        break;
      case STATE_ACCESS_DENIED:
        updateAccessDeniedScreen();
        break;
      case STATE_SYSTEM_LOCKED:
        updateSystemLockedScreen();
        break;
      case STATE_ULTRASONIC_ALERT:
        updateUltrasonicAlertScreen();
        break;
    }
  }

  // For states that need periodic updates
  if (currentLCDState == STATE_ACCESS_GRANTED && accessGranted) {
    unsigned long remainingTime = (ACCESS_DURATION - (millis() - accessTime)) / 1000;
    lcd.setCursor(0, 2);
    lcd.print(F("Time remaining: "));
    lcd.print(remainingTime);
    lcd.print(F("s  "));
    lcd.setCursor(0, 3);
    lcd.print(F("Door is UNLOCKED  "));
  }
  else if (currentLCDState == STATE_SYSTEM_LOCKED && systemLocked) {
    unsigned long remainingTime = (LOCKOUT_DURATION - (millis() - lockTime)) / 1000;
    lcd.setCursor(0, 2);
    lcd.print(F("Lockout time: "));
    lcd.print(remainingTime);
    lcd.print(F("s  "));
    lcd.setCursor(0, 3);
    lcd.print(F("Please wait...    "));
  }
  else if (currentLCDState == STATE_IDLE) {
    // Update time and sensor readings on idle screen
    static unsigned long lastTimeUpdate = 0;
    if (millis() - lastTimeUpdate > 1000) {
      lastTimeUpdate = millis();
      updateIdleScreenWithSensors();
    }
  }
  else if (currentLCDState == STATE_ULTRASONIC_ALERT) {
    // Update sensor readings on alert screen
    static unsigned long lastAlertUpdate = 0;
    if (millis() - lastAlertUpdate > 1000) {
      lastAlertUpdate = millis();
      updateUltrasonicAlertScreen();
    }
  }
}

void updateIdleScreen() {
  DateTime now = rtc.now();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(F("=== BANK SECURITY ==="));
  lcd.setCursor(0, 1);
  lcd.print(F("Enter 4-digit Code:"));
  lcd.setCursor(0, 2);
  lcd.print(F("Time: "));
  lcd.print(formatIndianTime(now));
  updateSensorDisplayLine();
}

void updateIdleScreenWithSensors() {
  DateTime now = rtc.now();
  lcd.setCursor(0, 2);
  lcd.print(F("Time: "));
  lcd.print(formatIndianTime(now));
  lcd.print(F("    "));
  updateSensorDisplayLine();
}

void updateSensorDisplayLine() {
  lcd.setCursor(0, 3);
  lcd.print(F("S1:"));
  if (distance1 < 100) lcd.print(F(" "));
  if (distance1 < 10) lcd.print(F(" "));
  lcd.print(distance1);
  lcd.print(F("cm S2:"));
  if (distance2 < 100) lcd.print(F(" "));
  if (distance2 < 10) lcd.print(F(" "));
  lcd.print(distance2);
  lcd.print(F("cm"));
}

void updateInputScreen() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(F("Password Input:"));
  lcd.setCursor(0, 1);
  lcd.print(F("Enter 4-digit code:"));
  lcd.setCursor(0, 2);
  lcd.print(F("Current: "));
  for (int i = 0; i < inputPassword.length(); i++) {
    lcd.print('*');
  }
  for (int i = inputPassword.length(); i < 4; i++) {
    lcd.print('_');
  }
  lcd.setCursor(0, 3);
  lcd.print(F("*=Clear  #=Submit  "));
  currentLCDState = STATE_INPUT;
}

void updateAccessGrantedScreen() {
  lcd.clear();
  lcd.setCursor(4, 0);
  lcd.print(F("ACCESS GRANTED"));
  lcd.setCursor(3, 1);
  lcd.print(F("Door Unlocked"));
  lcd.setCursor(0, 2);
  lcd.print(F("Time remaining: 10s"));
  lcd.setCursor(0, 3);
  lcd.print(F("Door is UNLOCKED  "));
  currentLCDState = STATE_ACCESS_GRANTED;
}

void updateAccessDeniedScreen() {
  lcd.clear();
  lcd.setCursor(4, 0);
  lcd.print(F("ACCESS DENIED"));
  lcd.setCursor(0, 1);
  lcd.print(F("Attempt: "));
  lcd.print(attemptCount);
  lcd.print(F("/"));
  lcd.print(MAX_ATTEMPTS);
  lcd.setCursor(0, 2);
  lcd.print(F("Invalid Password!"));
  lcd.setCursor(0, 3);
  lcd.print(F("Press * to retry   "));
  currentLCDState = STATE_ACCESS_DENIED;
}

void updateSystemLockedScreen() {
  lcd.clear();
  lcd.setCursor(3, 0);
  lcd.print(F("SYSTEM LOCKED"));
  lcd.setCursor(0, 1);
  lcd.print(F("Too many failed"));
  lcd.setCursor(0, 2);
  lcd.print(F("attempts!"));
  lcd.setCursor(0, 3);
  lcd.print(F("Please wait...    "));
  currentLCDState = STATE_SYSTEM_LOCKED;
}

void updateUltrasonicAlertScreen() {
  lcd.clear();
  lcd.setCursor(2, 0);
  lcd.print(F("SECURITY ALERT!"));
  lcd.setCursor(0, 1);
  lcd.print(F("Object detected"));
  lcd.setCursor(0, 2);
  lcd.print(F("too close!"));

  // Show which sensor detected the object with specific distances
  lcd.setCursor(0, 3);
  if (distance1 <= SENSOR1_ALERT_DISTANCE && distance2 <= HIGH_ALERT_DISTANCE) {
    lcd.print(F("Both sensors alert!"));
  } else if (distance1 <= SENSOR1_ALERT_DISTANCE) {
    lcd.print(F("S1 Alert:30cm!    "));
  } else {
    lcd.print(F("S2 Alert:20cm!    "));
  }
}

void displayAccessCountdown() {
  unsigned long remainingTime = (ACCESS_DURATION - (millis() - accessTime)) / 1000;
  Serial.print(F("Door will lock in: "));
  Serial.print(remainingTime);
  Serial.println(F(" seconds"));
}

void displayLockoutCountdown() {
  unsigned long remainingTime = (LOCKOUT_DURATION - (millis() - lockTime)) / 1000;
  Serial.print(F("System locked for: "));
  Serial.print(remainingTime);
  Serial.println(F(" seconds"));
}

// ========== PASSWORD FUNCTIONS ==========
String getCurrentPassword() {
  DateTime now = rtc.now();
  int currentMinute = now.minute();

  // Convert normal code to integer
  int codeValue = normalCode.toInt();

  // Add the current minute to the code
  int result = codeValue + currentMinute;

  // Format as 4-digit string (with leading zeros if needed)
  String resultStr = String(result);
  while (resultStr.length() < 4) {
    resultStr = "0" + resultStr;
  }

  // If result is more than 4 digits, take the last 4 digits
  if (resultStr.length() > 4) {
    resultStr = resultStr.substring(resultStr.length() - 4);
  }

  return resultStr;
}

void checkPassword() {
  DateTime now = rtc.now();
  String currentPassword = getCurrentPassword();

  Serial.println(F("Checking password..."));
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(F("Checking Password"));
  lcd.setCursor(0, 1);
  lcd.print(F("Please wait..."));
  if (inputPassword == currentPassword) {
    Serial.println(F("Access Granted - 10 seconds"));

    accessGranted = true;
    accessTime = millis();
    attemptCount = 0; // Reset attempts on success

    // Activate solenoid lock
    digitalWrite(SOLENOID_PIN, LOW);
    Serial.println(F("Solenoid unlocked"));

    updateAccessGrantedScreen();
    playAccessGrantedTone();
  }
  else {
    attemptCount++;
    Serial.println(F("Access Denied"));
    Serial.print(F("Attempts: "));
    Serial.print(attemptCount);
    Serial.print(F("/"));
    Serial.println(MAX_ATTEMPTS);

    updateAccessDeniedScreen();
    playAccessDeniedTone();

    // Check if system should be locked
    if (attemptCount >= MAX_ATTEMPTS) {
      systemLocked = true;
      lockTime = millis();
      Serial.println(F("SYSTEM LOCKED! Too many failed attempts."));
      Serial.println(F("Please wait for 10 seconds."));
      updateSystemLockedScreen();
      playSystemLockedTone();
    }
  }

  inputPassword = ""; // Reset after checking
}

String formatIndianTime(DateTime dt) {
  int hour = dt.hour();
  int minute = dt.minute();
  String period = "AM";

  // Convert to 12-hour format
  if (hour >= 12) {
    period = "PM";
    if (hour > 12) {
      hour -= 12;
    }
  }
  if (hour == 0) {
    hour = 12; // Midnight case
  }

  char buffer[9];
  sprintf(buffer, "%d:%02d %s", hour, minute, period.c_str());
  return String(buffer);
}

// ========== BUZZER FUNCTIONS ==========
void playStartupTone() {
  tone(buzzerPin, 1000, 100);
  delay(150);
  tone(buzzerPin, 1500, 100);
  delay(150);
  tone(buzzerPin, 2000, 100);
}

void playKeyTone() {
  tone(buzzerPin, 800, 50);
}

void playAccessGrantedTone() {
  tone(buzzerPin, 1500, 200);
  delay(250);
  tone(buzzerPin, 2000, 200);
  delay(250);
  tone(buzzerPin, 2500, 400);
}

void playAccessDeniedTone() {
  tone(buzzerPin, 300, 500);
  delay(300);
  tone(buzzerPin, 200, 500);
}

void playSystemLockedTone() {
  for (int i = 0; i < 3; i++) {
    tone(buzzerPin, 400, 200);
    delay(250);
    tone(buzzerPin, 300, 200);
    delay(250);
  }
}

void playSystemUnlockedTone() {
  tone(buzzerPin, 1500, 100);
  delay(120);
  tone(buzzerPin, 2000, 100);
  delay(120);
  tone(buzzerPin, 2500, 300);
}

void playAccessExpiredTone() {
  tone(buzzerPin, 400, 100);
  delay(150);
  tone(buzzerPin, 300, 100);
  delay(150);
  tone(buzzerPin, 200, 300);
}
```

6. WORKING OF THE SYSTEM

The system operates in two parallel modes: security monitoring and access control. Upon power-up, the system initializes the RTC and LCD, then begins continuous distance measurement using the ultrasonic sensor. The measured distance is categorized into zones: Green (safe), Warning, and Alert, triggering corresponding LED and buzzer patterns. Simultaneously, the system listens for input from the keypad. To gain access, a user must enter a 4-digit password, which is dynamically generated by combining a base code with the current minute value from the RTC. If the entered code matches, the solenoid lock is disengaged for 10 seconds. After three failed attempts, the system enters a lockout state. A push button allows toggling the ultrasonic monitoring on or off without affecting the access control function.

7. RESULT ANALYSIS

During execution, the system successfully reflects the following results: The LCD clearly displays the real-time status, including "Secure" when no object is close, "Warning: Object Approaching" when within 20-50 cm, and "Alert! Intrusion Detected" when within 20 cm, accompanied by the correct audio-visual signals. The access control system correctly validates passwords, granting access with a "Access Granted" message and unlocking the solenoid for exactly 10 seconds, followed by an "Auto-Relocking" message. After three incorrect password entries, the system displays "Lockout: Wait 10s" and temporarily disables the keypad, confirming the system works as intended per the design objectives.