Department of ECE

19ECE304 Microcontrollers and Interfacing Lab

Project Report- December 2024

Project Title: SMART KEYPAD ACCESS AND ATTENDANCE SYSTEM WITH LPC2148

Team Members (Name and Roll No)

AM.EN.U4ECE22023 - MANIYAR MAHAMMED AFNAN
AM.EN.U4ECE22026 - MURAKONDA CHANAKYA VENKATA KARTHEEK
AM.EN.U4ECE22027 - MURAPAKA SAI DURGA SURYA VENKATESH
AM.EN.U4ECE22040 - SAI VIRASITH MADDULA
AM.EN.U4ECE22056 - MAHADEVU VINEETH NARAYANA NAIDU

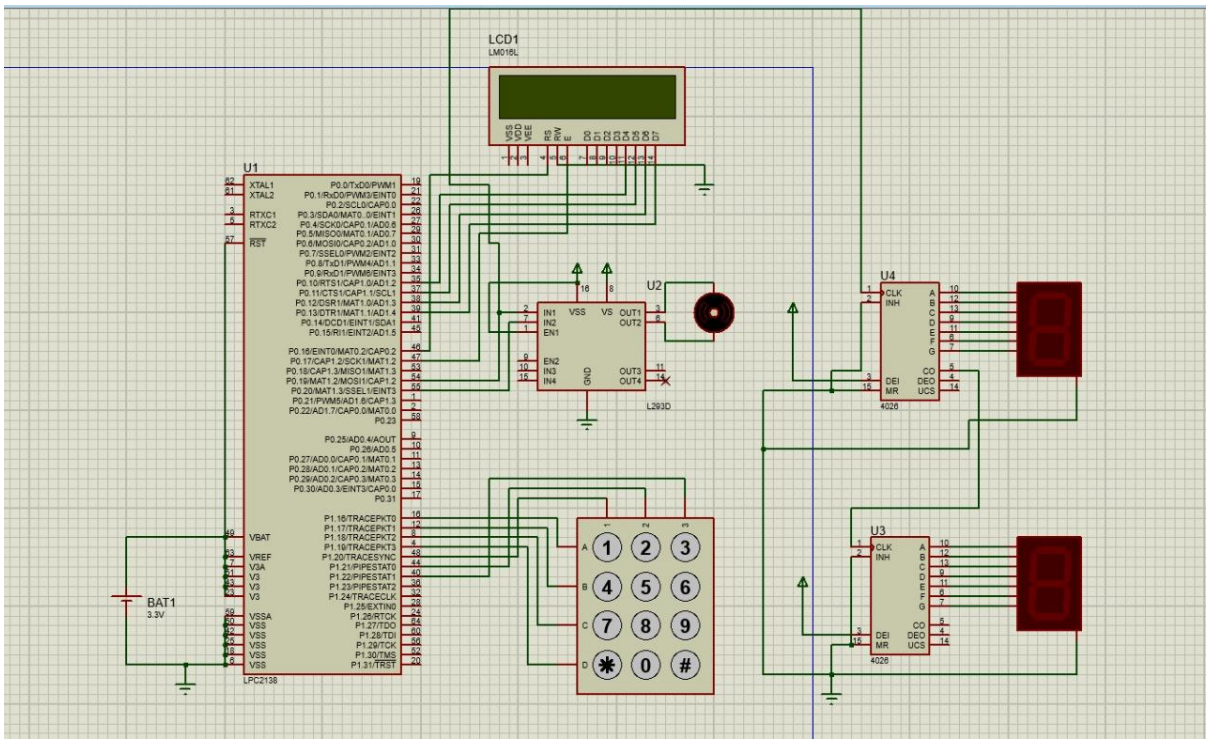
Signature of faculty with Date:

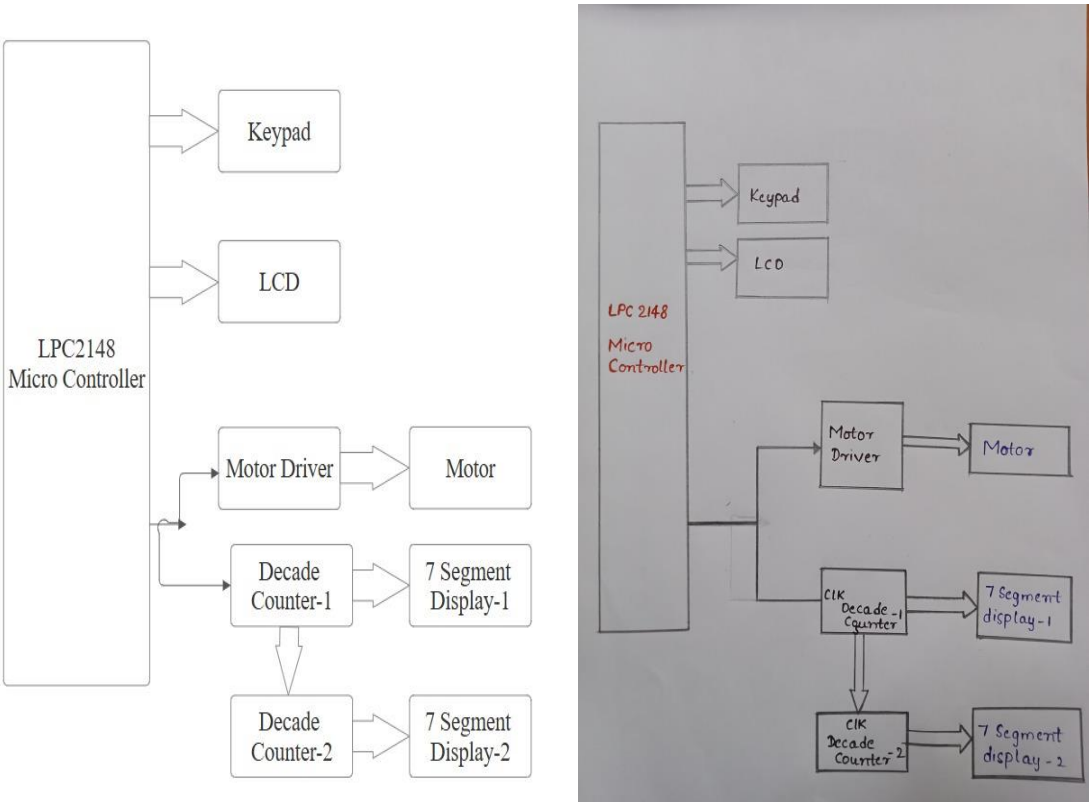
Project Title: **SMART KEYPAD ACCESS AND ATTENDANCE SYSTEM WITH LPC2148**

Objective: The objective of this project is to develop a secure electronic lock system using a microcontroller that operates based on a user-defined password entered via a keypad, with visual feedback provided on an LCD. The system allows for password updates and provides functionality for tracking attendance by logging successful password entries.

Components Required: LPC2148, Keypad, LCD, Motor Driver, Motor, Decade Counter, 7 Segment Display

Block Diagram/Circuit Diagram:





Theory: The project is designed to implement a secure electronic lock system using an LPC2148 microcontroller, which operates based on password authentication. The primary goal is to control access to a secure area by requiring a password, which is entered using a keypad. The system prompts the user for the password via an LCD display and compares the entered password with a predefined one. If the correct password is entered, the lock is activated, allowing access. The system also allows users to set a new password by entering #, enabling greater flexibility and security.

The project integrates several key components, including a keypad for user input, an LCD for user feedback, and a lock mechanism controlled by GPIO pins. The LPC2148 microcontroller serves as the central processing unit, managing the password input, comparison, and lock control. The lock mechanism is triggered using GPIO pins, which control a relay or motor to open or close the lock. Time delays are implemented through timer peripherals to ensure the lock remains in the correct state for the desired duration.

Additionally, the system offers a feature for attendance logging, where successful password entries are tracked. This could be expanded to monitor who enters a secure area and at what time, further enhancing the system's functionality. Overall, the project aims to provide a simple yet effective security solution with potential applications in areas requiring restricted access.

Description of Peripherals Used:

1. GPIO (General Purpose Input/Output):

- Keypad: GPIO pins are used to read the keypresses from the keypad in the get_key() function.

- LCD: GPIO pins are used to send commands and data to the LCD for displaying messages. The LCD control lines (RS, E, D4, D5, D6, D7) are connected to specific GPIO pins.

- Lock Control: GPIO pins 19 and 20 are used to control the opening and closing of the lock. Pin 19 is set high to open the lock and pin 20 is set high to close the lock.

2. Timers:

- timemdel() and timeudel() functions: Timers are used to create time delays. These delays control how long the lock stays open and closed. They also help manage the timing between commands for the LCD and keypad to ensure proper sequence execution.

Summary of Peripheral Usage:

- GPIO is used for interfacing with the keypad, controlling the LCD, and managing the lock mechanism.

- Timers are used to implement time delays in the lock control logic and for other timing-based operations.

Project Description/Working:

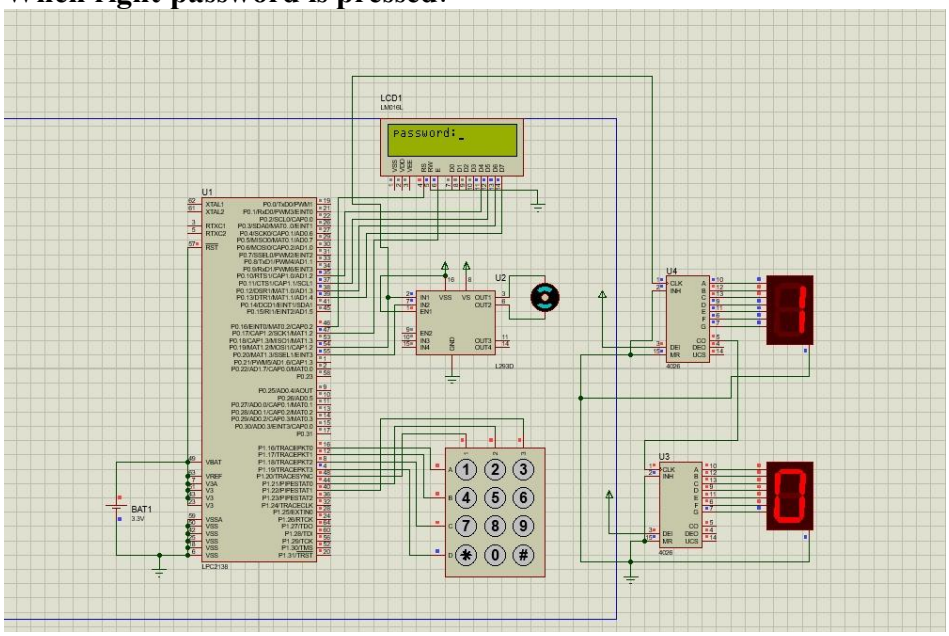
1. **Keypad:** The microcontroller interfaces with a keypad through GPIO pins. The keypad is connected to the microcontroller, and when a user enters the password, the microcontroller reads the key presses and processes them.
2. **LCD Display:** The microcontroller interfaces with an LCD display through GPIO pins. It sends commands and data to the LCD to display messages or prompts to the user. The LCD display provides visual feedback to the user, such as asking them to enter the password or displaying messages like "Access Granted" or "Access Denied."
3. **Motor Control:** The microcontroller controls the motor that opens and closes the door. It uses GPIO pins to interface with a motor driver or motor control circuit, which rotates the motor clockwise to open the door and anticlockwise to close it. The microcontroller triggers the motor rotation based on the correct password entry.
4. **Seven Segment Display:** The microcontroller interfaces with the seven-segment display through GPIO pins. It increments the seven-segment display to keep track of the number of times the door is opened successfully, indicating attendance tracking.
5. **Password Verification:** When the user enters the password through the keypad, the microcontroller reads the input, compares it with the pre-set correct password, and

verifies if it matches. If the entered password is correct, the microcontroller activates the motor to open the door.

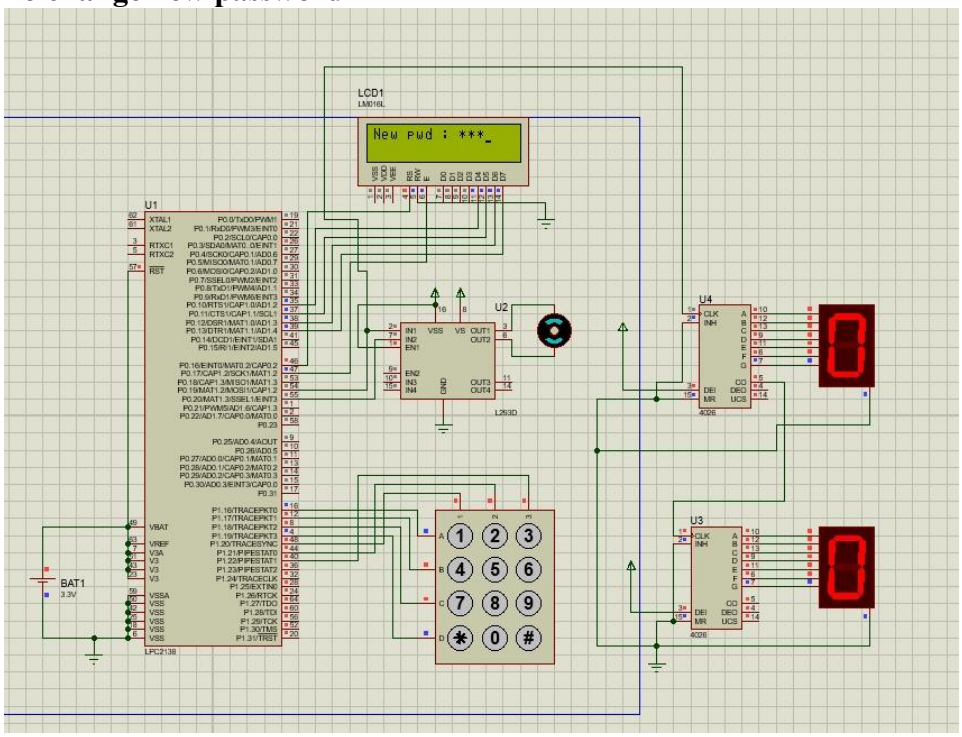
6. **Attendance Tracking:** After successful verification of the password and opening the door, the microcontroller increments the seven-segment display to track the number of times the door is opened successfully. This feature provides automatic attendance tracking for each time the door is accessed.

Result/Output: The output of this project is a fully operational password-protected electronic lock system with attendance tracking capabilities. When the correct password is entered on the keypad, the LCD displays a message indicating that the lock is being unlocked, and the lock mechanism is activated. After a set duration, the lock automatically closes, ensuring security. If an incorrect password is entered, the LCD displays an error message, and the lock remains closed. Additionally, each successful password entry is logged for attendance tracking, allowing the system to monitor access. Real-time feedback through the LCD ensures seamless operation, enhanced security, and the ability to track user activity effectively.

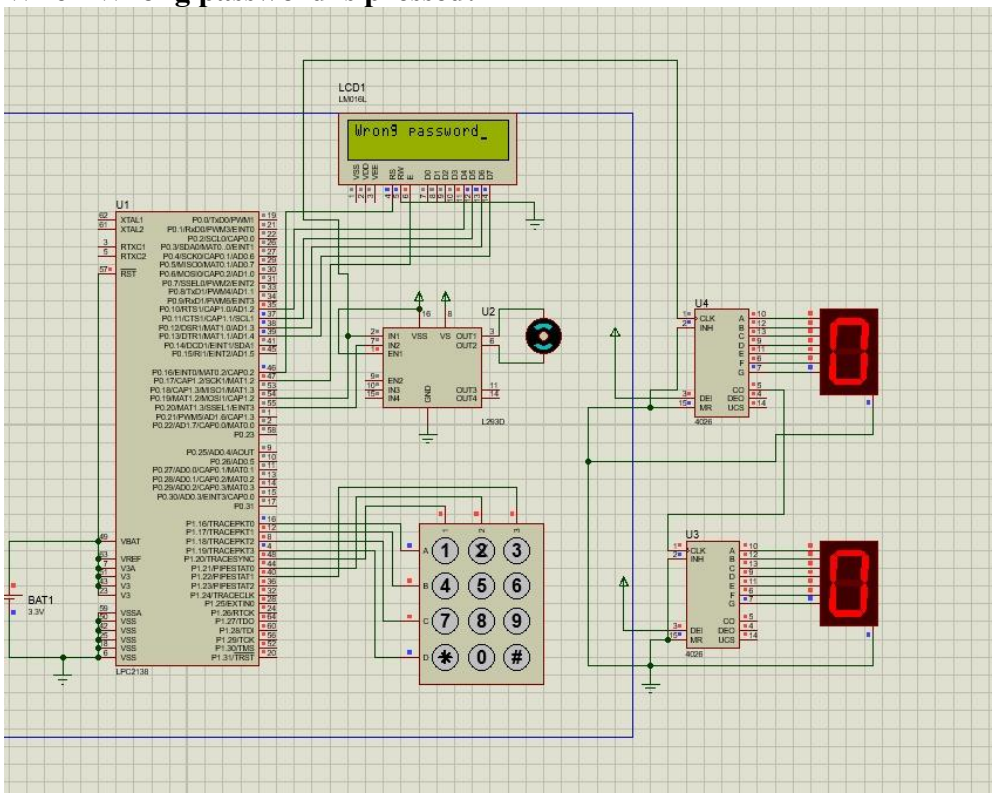
When right password is pressed:



To change new password:



When Wrong password is pressed:



Conclusion: In conclusion, this project successfully implements a secure, password-protected electronic lock system using an LPC2148 microcontroller. The system provides reliable access control by allowing users to set and authenticate passwords via a keypad, with real-time feedback displayed on an LCD. The integration of attendance tracking enhances the system's functionality, enabling monitoring of access logs. By combining user-friendly interaction, robust security features, and real-time operation, this project demonstrates a practical and scalable solution for securing restricted areas.

References:

1) https://github.com/a3X3k/Password-Based-Door-Open-System-Using-LPC2148

2) https://github.com/pbnithin1/PASSWORD-BASED-DOOR-OPEN-SYSTEM-USING-LPC2148

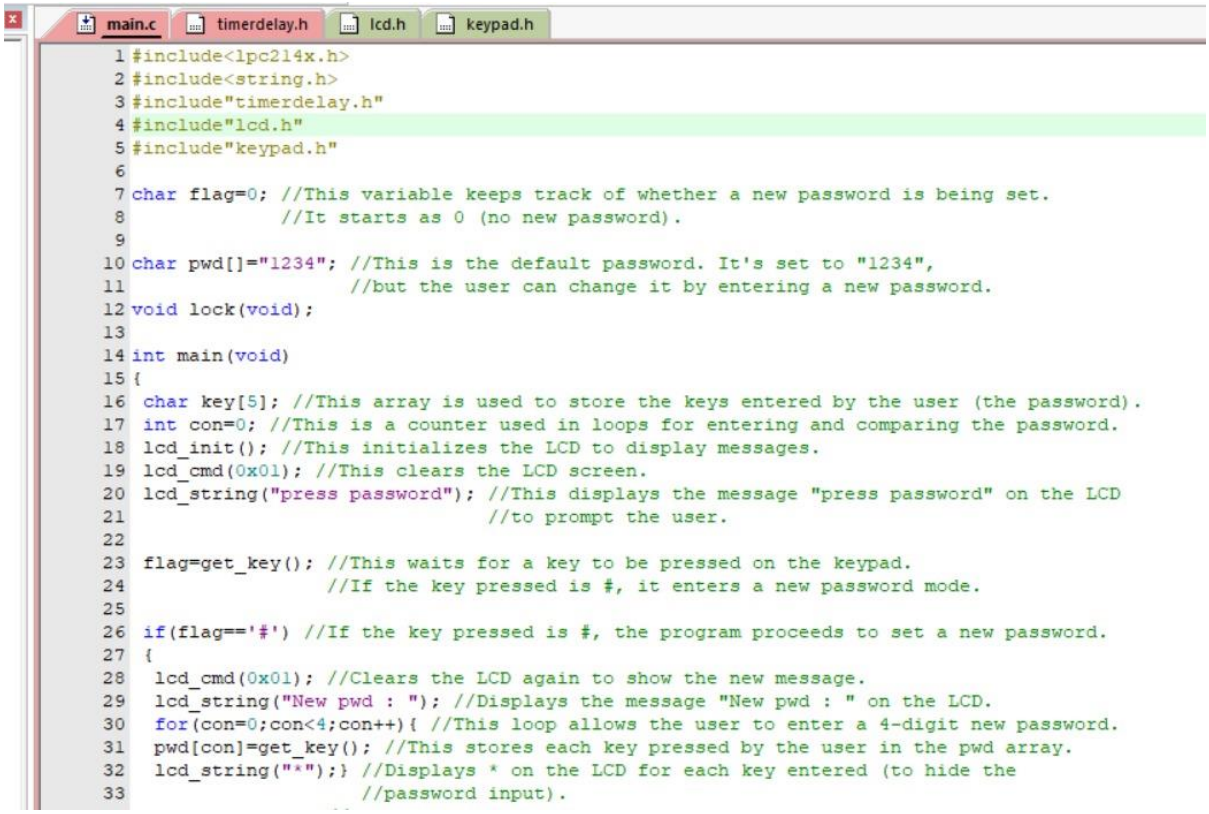
3) https://embetronicx.com/projects/lpc2148-projects/password-based-door-open-system-using-lpc2148/

These project serves as a reference for our project, providing a foundational concept for implementing a password-based electronic lock system using the LPC2148 microcontroller. While the core functionality of password authentication and lock control is similar, our project goes a step further by integrating additional features, such as attendance tracking. This enhancement allows us to monitor and log successful password entries, adding an extra layer of functionality beyond the basic lock mechanism.

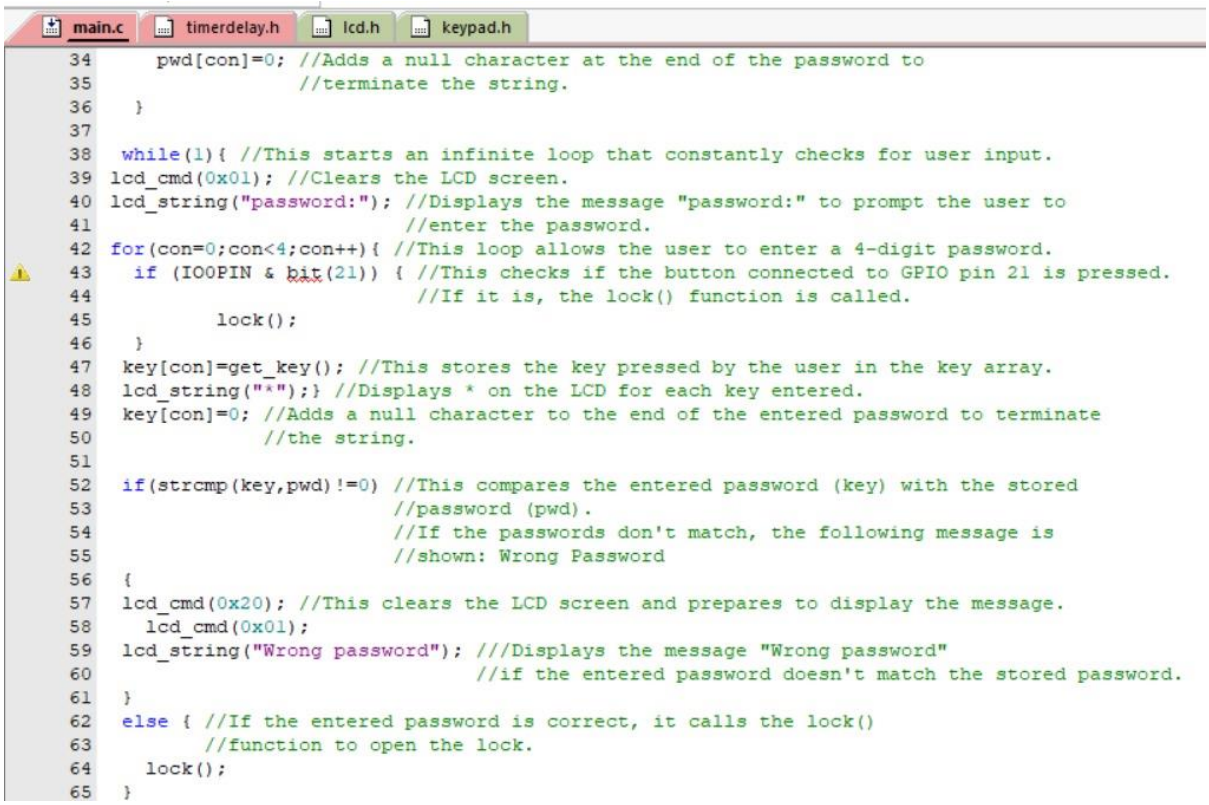
Acknowledgement: I would like to extend my heartfelt gratitude to my mentors and faculty members for their continuous guidance and encouragement throughout this project. Their expertise and support were instrumental in the successful completion of this work. I also thank my peers for their valuable input and teamwork, which greatly enhanced the quality of the project.

Appendix: Code with Comments

Main.c:



```
#include<lpc214x.h>
#include<string.h>
#include"timerdelay.h"
#include"lcd.h"
#include"keypad.h"

char flag=0; //This variable keeps track of whether a new password is being set.
             //It starts as 0 (no new password).

char pwd[]="1234"; //This is the default password. It's set to "1234",
                   //but the user can change it by entering a new password.
void lock(void);

int main(void)
{
 char key[5]; //This array is used to store the keys entered by the user (the password).
 int con=0; //This is a counter used in loops for entering and comparing the password.
 lcd_init(); //This initializes the LCD to display messages.
 lcd_cmd(0x01); //This clears the LCD screen.
 lcd_string("press password"); //This displays the message "press password" on the LCD
                               //to prompt the user.

 flag=get_key(); //This waits for a key to be pressed on the keypad.
                 //If the key pressed is #, it enters a new password mode.

 if(flag=='#') //If the key pressed is #, the program proceeds to set a new password.
 {
  lcd_cmd(0x01); //Clears the LCD again to show the new message.
  lcd_string("New pwd : "); //Displays the message "New pwd : " on the LCD.
  for(con=0;con<4;con++){ //This loop allows the user to enter a 4-digit new password.
  pwd[con]=get_key(); //This stores each key pressed by the user in the pwd array.
  lcd_string("*");} //Displays * on the LCD for each key entered (to hide the
                    //password input).
```



```
     pwd[con]=0; //Adds a null character at the end of the password to
                 //terminate the string.
   }

 while(1){ //This starts an infinite loop that constantly checks for user input.
lcd_cmd(0x01); //Clears the LCD screen.
lcd_string("password:"); //Displays the message "password:" to prompt the user to
                         //enter the password.
for(con=0;con<4;con++){ //This loop allows the user to enter a 4-digit password.
  if (IO0PIN & bit(21)) { //This checks if the button connected to GPIO pin 21 is pressed.
                          //If it is, the lock() function is called.
        lock();
  }
 key[con]=get_key(); //This stores the key pressed by the user in the key array.
 lcd_string("*");} //Displays * on the LCD for each key entered.
 key[con]=0; //Adds a null character to the end of the entered password to terminate
             //the string.

 if(strcmp(key,pwd)!=0) //This compares the entered password (key) with the stored
                        //password (pwd).
                        //If the passwords don't match, the following message is
                        //shown: Wrong Password
 {
 lcd_cmd(0x20); //This clears the LCD screen and prepares to display the message.
   lcd_cmd(0x01);
 lcd_string("Wrong password"); ///Displays the message "Wrong password"
                               //if the entered password doesn't match the stored password.
 }
 else { //If the entered password is correct, it calls the lock()
        //function to open the lock.
   lock();
 }
```

```
}
}


void lock(void)
{
 IO0DIR|=bit(19)|bit(20); //Sets GPIO pins 19 and 20 as output pins.
                          //These pins control the lock mechanism.

 lcd_cmd(0x01); //Clears the LCD screen.
 lcd_string("lock is opening"); //Displays the message "lock is opening".
 IO0SET|=bit(19); //Sets GPIO pin 19 high, activating the mechanism to open the lock.
 timemdel(50); //Creates a small delay using the timemdel function to keep the
               //lock open for a short time.

 IO0CLR|=bit(19); //Sets GPIO pin 19 low, deactivating the lock.
 timemdel(100); //Creates a longer delay.
 lcd_cmd(0x01); //Clears the LCD again.
 lcd_string("lock is closing"); //Displays the message "lock is closing".
 IO0SET|=bit(20); //Sets GPIO pin 20 high, activating the mechanism to close the lock.
 timemdel(50); //Creates a small delay.
 IO0CLR|=bit(20); //Sets GPIO pin 20 low, deactivating the lock.
}
```

Lcd.h :

main.c | timerdelay.h | lcd.h | keypad.h

```
/*----LCD Header File------*/
#define RS (1<<16)  //RS (Register Select): Pin 16
#define E (1<<17)   //E (Enable): Pin 17
#define D4 (1<<10)  //D4, D5, D6, D7: Data pins for the LCD (Pins 10, 11, 12, 13)
#define D5 (1<<11)
#define D6 (1<<12)
#define D7 (1<<13)


//void delay(void);
void lcd_init(void);
void lcd_cmd(int );
void lcd_string(char *str);
void lcd_conv(char );


void lcd_cmd(int cmd) //This function sends a command to the LCD (e.g., clear display, set cursor position).
{
 IOCLR0|=RS;  //Clears the RS pin (sets it to 0). This is necessary because RS should be low
              //for sending commands to the LCD.

 lcd_conv(cmd); // Calls the lcd_conv function to send the command.
 IOCLR0|=RS;
 timeudel(1000); //This function creates a delay to give the LCD enough time to
                 //process the command (typically a microsecond delay).
}
```

main.c | timerdelay.h | lcd.h | keypad.h

```
void lcd_init(void) //This function initializes the LCD
{
 IODIR0|=D4|D5|D6|D7; // Configures the data pins D4, D5, D6, D7 as output.
  IODIR0|=RS|E; //Configures the control pins RS and E as output.
  IOCLR0|=D4|D5|D6|D7; //Clears (sets to 0) the data pins D4 to D7.
  IOCLR0|=RS|E; // Clears the control pins RS and E.
 lcd_cmd(0x02); //Initializes the LCD in 4-bit mode.
 lcd_cmd(0x28); //Sets the LCD to use 2 lines and 5x8 font.
 lcd_cmd(0x0E); //Turns the display on and enables the cursor.
 lcd_cmd(0x01); // Clears the display.
 lcd_cmd(0x06); //Sets the cursor to move to the right automatically.
}


void lcd_string(char *str) //This function displays a string on the LCD
{
 while(*str!=0) //Loops through each character in the string str until the null-terminator (\0) is reached.
 {
  IOSET0|=RS; //Sets the RS pin high, which tells the LCD to interpret the data as characters.
  lcd_conv(*str); //Converts the current character (*str) into its 4-bit representation and
                  //sends it to the LCD.
  str++; //Moves to the next character in the string.
 }
 timeudel(1000); //Creates a small delay after the entire string is displayed.
}
```

```c
void lcd_conv(char data) //This function converts the character data into two 4-bit nibbles (
                         //high and low) and sends them to the LCD
{
 int recv;
 IOCLR0|=D4|D5|D6|D7;
 recv=data>>4; //Takes the high nibble (the top 4 bits) of the character data.
 IOSET0=recv<<10; //Sends the high nibble to the LCD via data pins D4 to D7.
 IOSET0|=E; //Enables the LCD to read the data.
 timeudel(1000); //Waits for the LCD to process the data.
 IOCLR0|=E; //Disables the LCD after sending the data.
 timeudel(1000);

 IOCLR0|=D4|D5|D6|D7;
 recv=data; //for low nibble(no need for shifting in this case as it's already the low nibble).
 IOSET0|=(recv<<10);
 IOSET0|=E;
 timeudel(1000);
 IOCLR0|=E;
 IOCLR0|=D4|D5|D6|D7;
}
```

Keypad.h :

main.c | timerdelay.h | lcd.h | keypad.h

```c
/*----Keypad Header File------*/
#define c1 (1<<20)  //c1, c2, c3 represent the columns of the keypad (connected to pins 20, 21, 22).
#define c2 (1<<21)
#define c3 (1<<22)
#define r1 (1<<16)  //r1, r2, r3, r4 represent the rows of the keypad (connected to pins 16, 17, 18, 19).
#define r2 (1<<17)
#define r3 (1<<18)
#define r4 (1<<19)

char get_key(void)  //This function will scan the keypad and return the key pressed by the user.
{
 IO1DIR=0x0f<<16;  //This sets the direction of the pins for rows (pins 16-19).
                   //It configures the first 4 pins (16-19) to output (for rows) and leaves the
                   //rest of the pins as input.
                   //IO1DIR is the direction register of Port 1.

  IO1CLR|=0xFF<<16;  //This clears the output pins connected to the rows (pins 16-19), setting them to low.
 IO1SET|=0xFF<<16;  //This sets the output pins connected to the rows (pins 16-19) to high.


 while(1)  //This is an infinite loop to continuously scan the keypad until a key is pressed.
 {


  IO1CLR|=r1; //IO1CLR|=r1 clears (sets low) the row r1 (pin 16).
 IO1SET|=0x0E<16; //IO1SET|=0x0E<<16 sets the columns c1, c2, c3 (pins 20, 21, 22) to high.
  if((IO1PIN&c1)==0)  //(IO1PIN&c1)==0 checks if the c1 (pin 20) is low (meaning the key is pressed).
                      //If yes, it returns the key '1'.
  {
    while((IO1PIN&c1)==0);
 return '1';
  }
  else if((IO1PIN&c2)==0) //Similarly, if c2 or c3 (pins 21 or 22) are low, it returns '2' or '3'.
  {
    while((IO1PIN&c2)==0);
 return '2';
  }
  else if((IO1PIN&c3)==0)
  {
    while((IO1PIN&c3)==0);
 return '3';
  }

```

```
//This block checks the columns to return '4', '5', or '6' depending on which column is pressed.
    IO1CLR|=r2;  //IO1CLR|=r2 clears row r2 (pin 17).
  IO1SET|=0x0D<<16;  //IO1SET|=0x0D<<16 sets the columns c1, c2, c3 (pins 20, 21, 22) to high.
    if((IO1PIN&c1)==0)
   {
     while((IO1PIN&c1)==0);
  return '4';
   }
  else if((IO1PIN&c2)==0)
   {
     while((IO1PIN&c2)==0);
  return '5';
   }
  else if((IO1PIN&c3)==0)
   {
     while((IO1PIN&c3)==0);
  return '6';
   }


//This block checks the columns to return '7', '8', or '9' depending on which column is pressed.
    IO1CLR|=r3;  //IO1CLR|=r3 clears row r3 (pin 18).
   IO1SET|=0X0B<<16; //IO1SET|=0x0B<<16 sets the columns c1, c2, c3 (pins 20, 21, 22) to high.
    if((IO1PIN&c1)==0)
   {
     while((IO1PIN&c1)==0);
  return '7';
   }
  else if((IO1PIN&c2)==0)
   {
     while((IO1PIN&c2)==0);
  return '8';
   }
  else if((IO1PIN&c3)==0)
   {
     while((IO1PIN&c3)==0);
  return '9';
   }


//This block checks the columns to return '*', '0', or '#' depending on which column is pressed.
  IO1CLR|=r4; //IO1CLR|=r4 clears row r4 (pin 19).
   IO1SET|=0X07<<16; //IO1SET|=0x07<<16 sets the columns c1, c2, c3 (pins 20, 21, 22) to high.
       if((IO1PIN&c1)==0)
   {
     while((IO1PIN&c1)==0);
  return '*';
   }
  else if((IO1PIN&c2)==0)
   {
     while((IO1PIN&c2)==0);
  return '0';
   }
  else if((IO1PIN&c3)==0)
   {
     while((IO1PIN&c3)==0);
  return '#';
   }
 }
}
```

Timerdelay.h :

main.c | timerdelay.h | lcd.h | keypad.h

```
1
2  /*---- Timerdelay Header File ----*/
3  void timemdel(unsigned int con) {
4      T0CTCR = 0x0; //This sets the Timer 0 Control Register (T0CTCR) to 0x0, which configures Timer 0 to
5                    //use the default timer mode (counting the clock).
6                    //This ensures Timer 0 operates normally (not in external event mode or other special modes).
7
8      T0PR = 59999; //This sets the Timer 0 Prescale Register (T0PR) to 59999.
9                    //This means that Timer 0 will count from 0 to 59999 before triggering an event.
10                   //Essentially, the prescaler divides the system clock frequency by 60,000.
11                   //If the system clock is 60 MHz, this would create a 1 kHz timer (1 ms time period).
12
13     T0MR0 = con; //This sets the Match Register 0 (T0MR0) to the value con passed to the function.
14                  //T0MR0 determines the timer's countdown limit.
15                  //The timer will count from 0 up to the value in T0MR0,
16                  //and when it reaches that value, it will trigger a match event.
17
18     T0MCR |= 1 << 2; //This modifies the Match Control Register (T0MCR), specifically setting the
19                      //third bit (bit 2) to 1.
20                      //This tells the timer to reset the timer when the match occurs (when the timer
21                      //reaches the value in T0MR0).
22
23     T0TCR = 0x02; //This sets the Timer Control Register (T0TCR) to 0x02, which stops the timer.
24                   //This is a reset step to make sure the timer is not running before starting it again.
25
26     T0TCR = 0x01; //This sets the Timer Control Register (T0TCR) to 0x01, which starts the timer.
27                   //This begins counting from 0, and it will trigger a match when it reaches the value in T0MR0.
28
29     while (T0TC != T0MR0); //This line waits in a loop until the Timer 0 Counter Register (T0TC)
30                            //equals the value in T0MR0.
31                            //Essentially, the program will pause here until the timer finishes counting.
32
33     T0TC = 0; //This clears the Timer 0 Counter Register (T0TC), resetting the timer counter to 0.
34
```

main.c | timerdelay.h | lcd.h | keypad.h

```
34
35     T0TCR = 0; //This stops the timer by setting the Timer Control Register (T0TCR) to 0.
36               //The timer is now off and ready to be used again if needed.
37 }
38
39 void timeudel(unsigned int con) { //This function works in the same way as the timemdel function
40                                   //but uses Timer 1 instead of Timer 0.
41
42     T1CTCR = 0x0; //This sets the Timer 1 Control Register (T1CTCR) to 0x0, ensuring that
43                   //Timer 1 works in normal mode (not in external event mode).
44
45     T1PR = 59; //This sets the Timer 1 Prescale Register (T1PR) to 59.
46                //The timer will count from 0 to 59 before triggering a match.
47                //This makes the timer run at 1 MHz if the system clock is 60 MHz (this is similar to Timer 0).
48
49     T1MR0 = con; //This sets the Match Register 0 (T1MR0) to the value of con passed to the function.
50                  //This determines when the timer will match and stop counting.
51
52     T1MCR |= 1 << 2; //This sets bit 2 of the Match Control Register (T1MCR), which causes the
53                      //timer to reset when it matches the value in T1MR0.
54
55     T1TCR = 0x02; //This stops Timer 1 by writing 0x02 to the Timer Control Register (T1TCR).
56
57     T1TCR = 0x01; //This starts Timer 1 by writing 0x01 to the Timer Control Register (T1TCR).
58
59     while (T1TC != T1MR0); //This line makes the program wait in a loop until the timer reaches
60                            //the match value (the value in T1MR0).
61
62     T1TC = 0; //This resets the Timer 1 Counter Register (T1TC) to 0, effectively clearing the counter.
63
64     T1TCR = 0; //This stops Timer 1 by setting the Timer Control Register (T1TCR) to 0.
65 }
```