Project 1 - Digital Lockbox

Tanvi Kharkar
EE 329-05
Spring Quarter 2021
Professor Hummel

Device Behavior Description

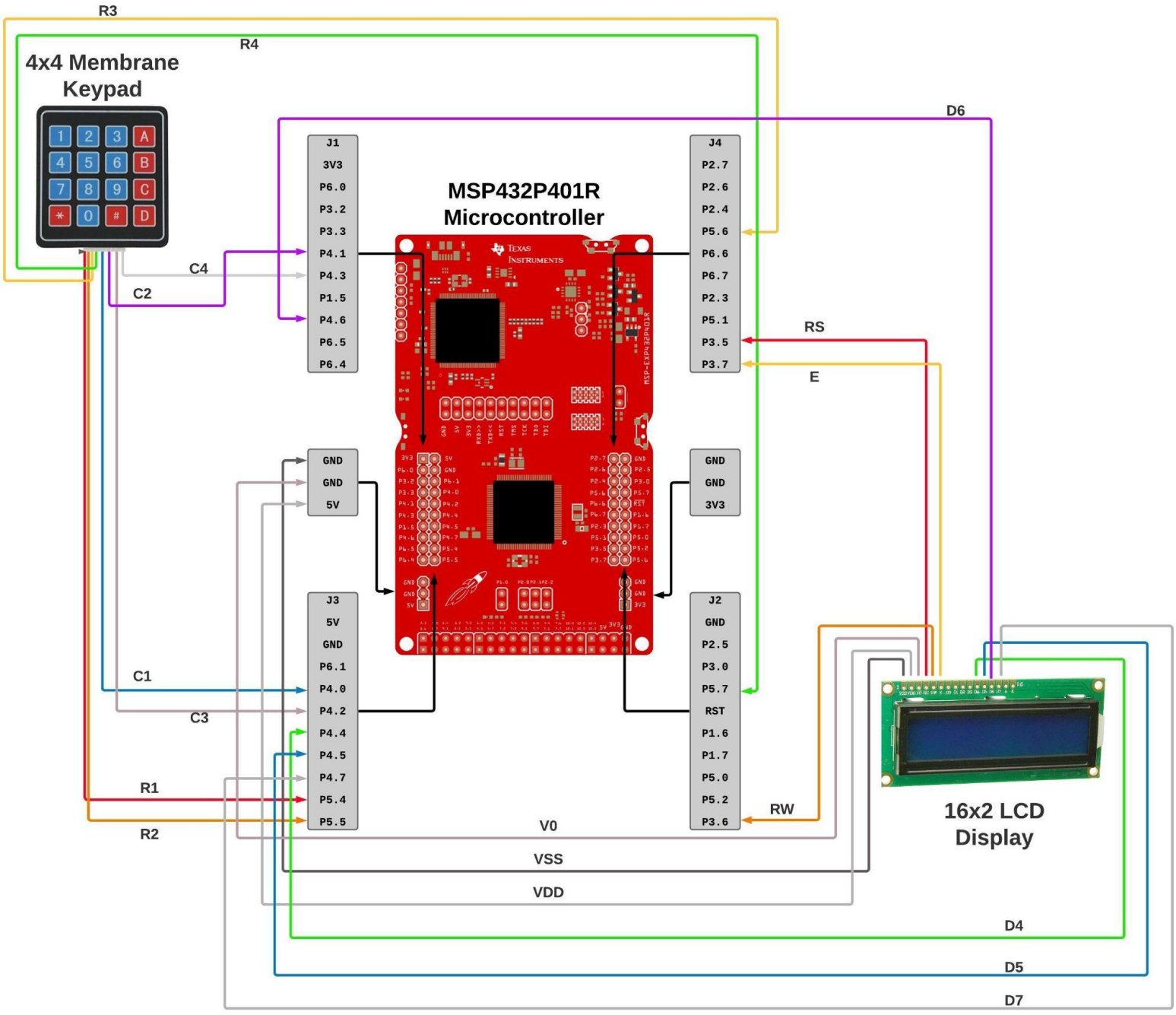
This project designs the MSP432P401R microcontroller in conjunction with a 4x4 membrane keypad and a 16x2 LCD screen as a digital lockbox. The lockbox starts with a default set pin code which, if entered correctly, will “unlock” the lockbox and give the user an opportunity to either relock the lockbox with the same password or set a new pin with which they can use to re-unlock the lockbox. When the lockbox is locked or in the process of locking, the RGB LED will be red and when it is unlocked, the RGB LED will be green.

System Specification

System Parameter	System Specification
Clock Frequency	3MHz
Power Supply Voltage	1.62V to 3.3V
Power Consumption (LCD Screen + Backlight)	0.5W
Display Size	16 x 2
View Angle	-20° to 35°
Optical Response Time	250ms

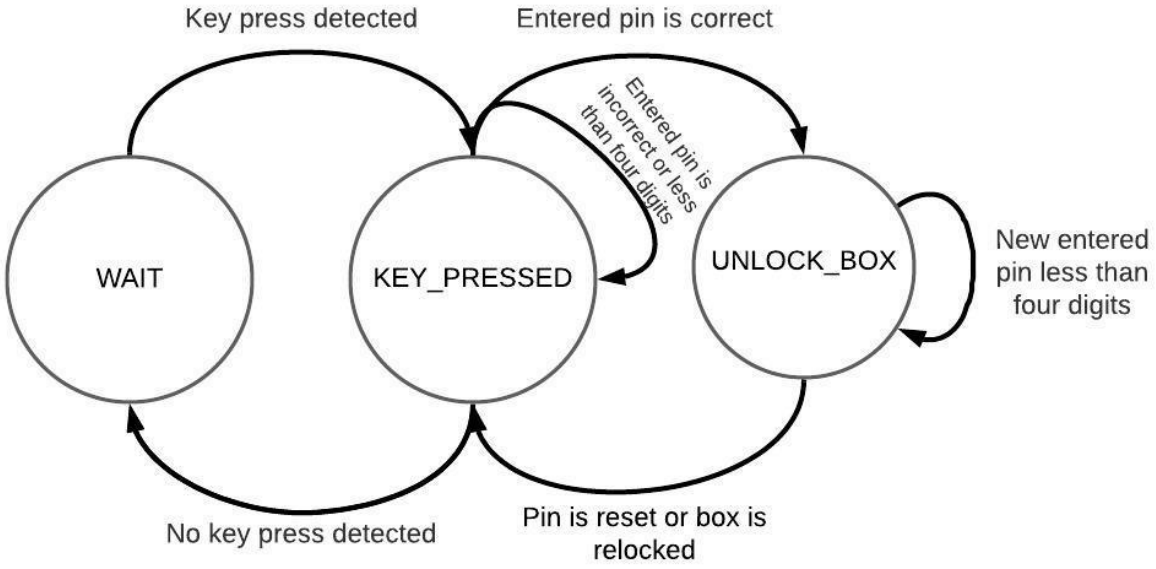
System Schematic

*Note: R and C represent “Row Signal” and “Column Signal” respectively.

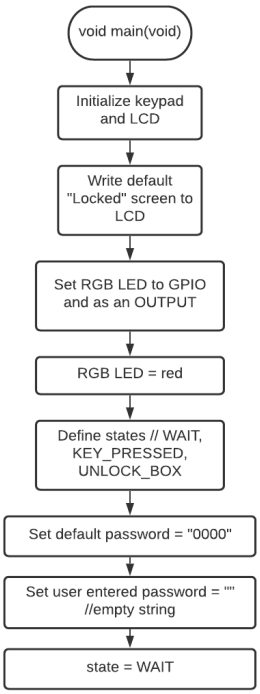


Software Architecture

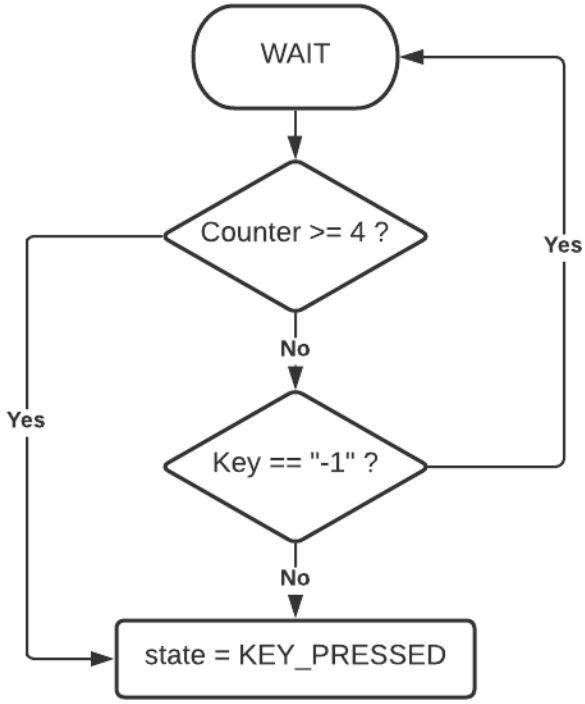
State Diagram



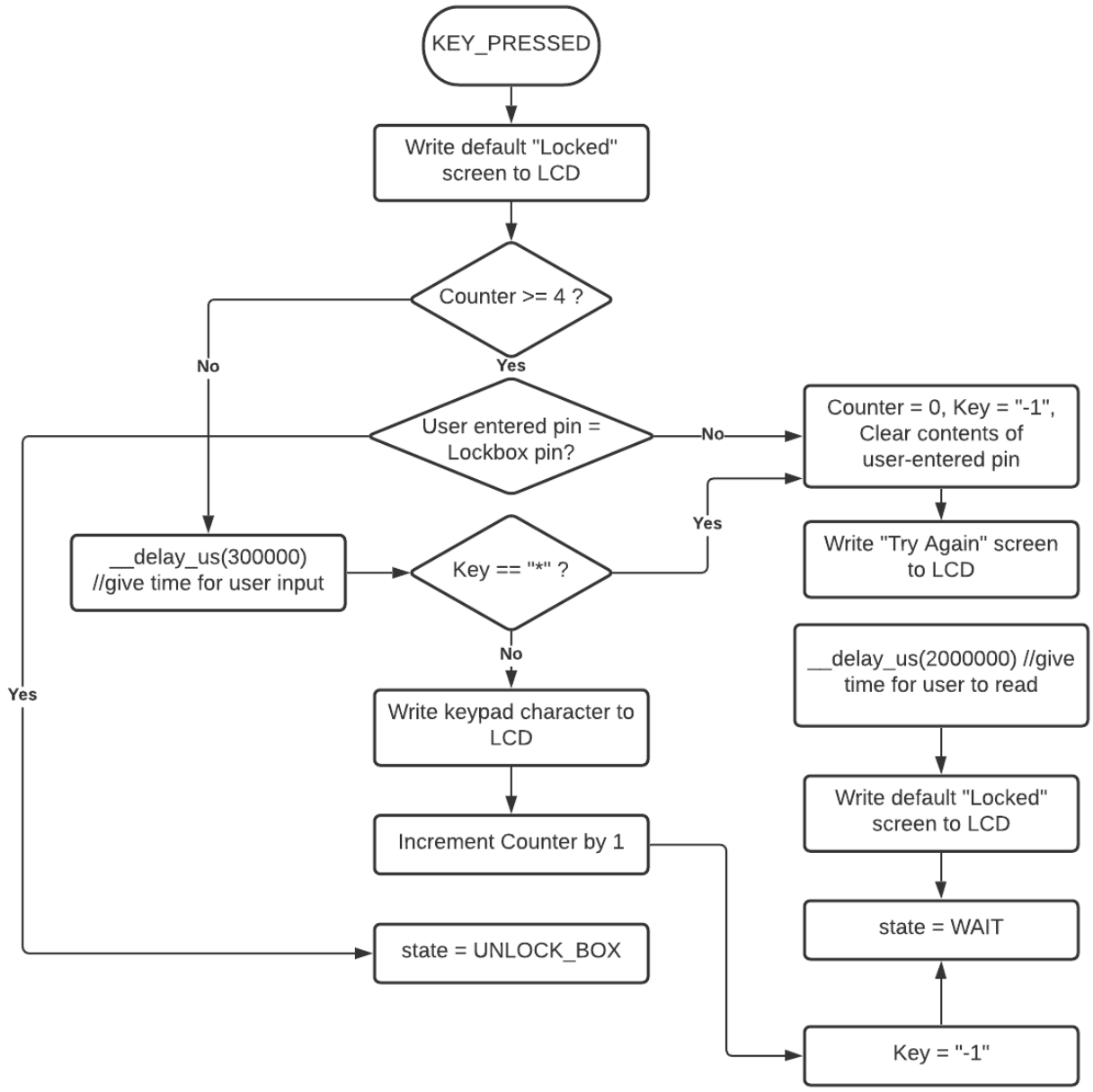
Main Diagram



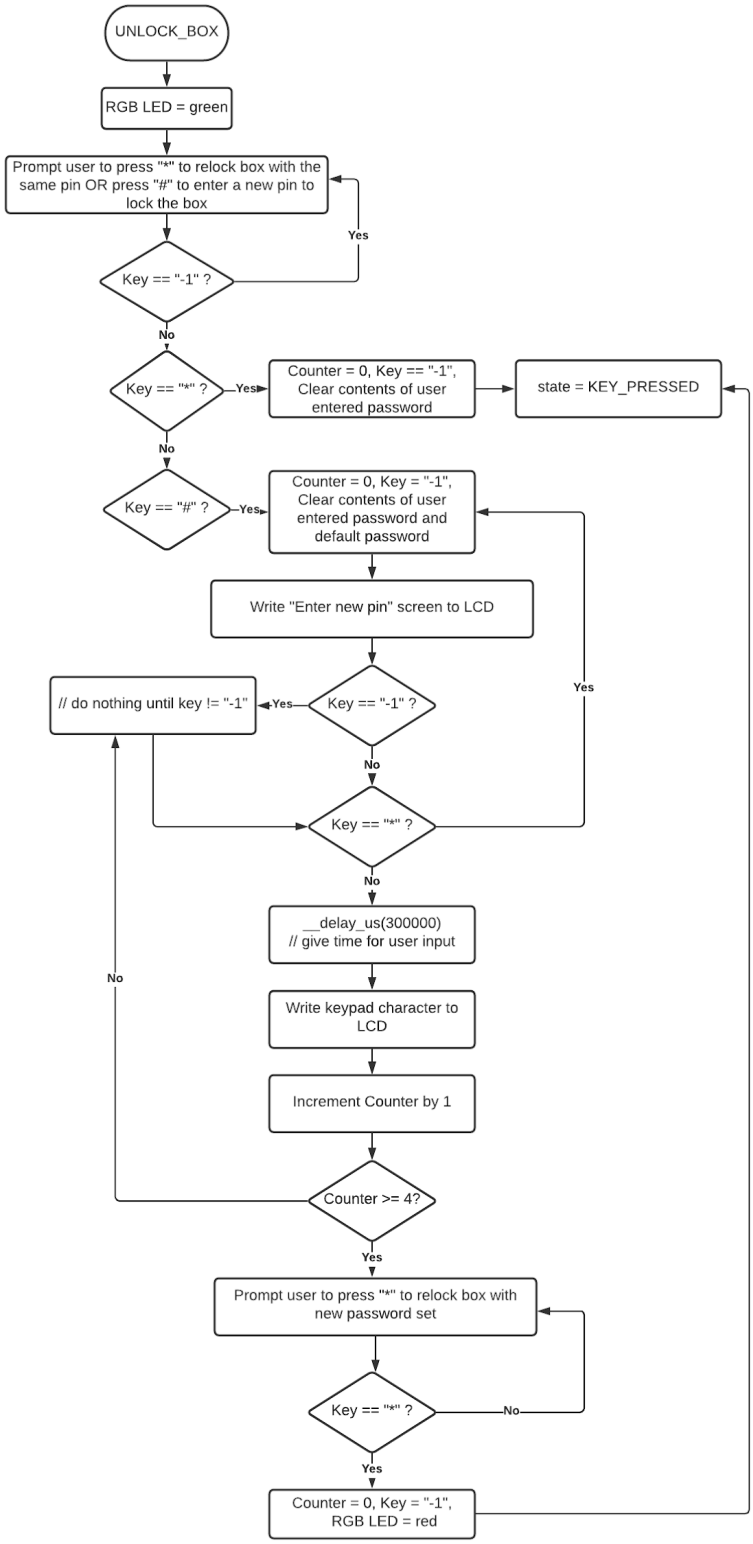
WAIT State Diagram



KEY_PRESSED State Diagram



UNLOCK_BOX State Diagram



Appendices

main.c

```
#include "msp.h"
#include <stdio.h>
#include "LCD.h"
#include "Keypad.h"
#include "string.h"

void main(void)
{
   WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;    // stop watchdog timer

   LCD_init(); // Initialize LCD
   keypad_init(); // Initialize keypad

    LCD_write_string("LOCKED"); // Set initial LCD display screen
    LCD_command(LINE_TWO);
    LCD_write_string("ENTER KEY: ");

    P2->SEL0 &= ~RGB_LED; // Initialize RGB LED to GPIO and as an OUTPUT
    P2->SEL1 &= ~RGB_LED;
    P2->DIR |= RGB_LED;
    P2->OUT = RED; // Set RGB LED to red

    char* key = "-1"; // Key "not pressed" value
    char counter = 0;
    char password[5] = "0000"; // Initial default password
    char new_password[5] = "";

    typedef enum // Define FSM States
     {
         WAIT,
         KEY_PRESSED,
         UNLOCK_BOX
     } STATE_TYPE;

     STATE_TYPE state = WAIT; // Start at WAIT state

     while(1)
     {
         switch(state) // Switch statement to cycle through FSM cases
         {
             case WAIT:
                 while (strcmp(key, "-1") == 0) // Wait until key press detected
                 {
                     key = getkey(); // Gets key from keypad press
                     if (counter >= 4) // Checks if more than 4 digits (length of pin code) are entered
                     {
                         state = KEY_PRESSED; // Go to KEY_PRESSED state to check if the entered pin is correct
                         break;
                     }
                 }


             case KEY_PRESSED:
                 if (strcmp(password, new_password) != 0) // Check if passwords are the same
                 {                       // If passwords are not the same, start over
                     if (counter >= 4) // Check if more than 4 incorrect digits are entered
```

```
        {
          __delay_us(USER_DELAY);
            LCD_init();  // Try again, reset screen
            LCD_write_string("INCORRECT KEY");
            LCD_command(LINE_TWO);
            LCD_write_string("TRY AGAIN");
          __delay_us(USER_READ);
            LCD_init();
            LCD_write_string("LOCKED"); // User must enter new password to be checked, process starts over
            LCD_command(LINE_TWO);
            LCD_write_string("ENTER KEY: ");
          key = "-1";
          counter = 0;
          memset(new_password,0,strlen(new_password));    // Clears the user entered password string
          state = WAIT;                                   // Wait for a key press
        }

      while (counter < 4)  // Check if four or less keys are entered by the user
      {
          if (strcmp(key, "*") == 0) // Clear LCD display if "*" key is pressed
          {
              LCD_init();
              LCD_write_string("LOCKED");
              LCD_command(LINE_TWO);
              LCD_write_string("ENTER KEY: ");
              counter = 0;
              key = "-1";
              memset(new_password,0,strlen(new_password));  // Clears the user entered password string
              state = WAIT;  // Wait for a key press
          }

          else if (strcmp(key, "*") != 0)  // Write keypad character to LCD display if not a "*"
          {
              if (strcmp(key, "-1") != 0)  // Check if any key is pressed
              {
                  __delay_us(USER_DELAY);  // Delay writing to LCD to give time for user input
                  LCD_write_string(key);   // Write keypad character to LCD screen
                  counter = counter + 1;   // Increment counter to check if entered pin is < 4 digits
                  strcat(new_password, key); // Formation of the user-entered password
                  key = "-1";  // Set key to default "not pressed" value
                  state = WAIT;  // Wait for a key press
              }
          }
          break;
      }
      break;
    }

case UNLOCK_BOX:
    __delay_us(USER_DELAY);
    P2->OUT = GREEN;  // Set LED to green when the box is unlocked
    LCD_init();  // LCD_init() is used to clear the LCD screen
    LCD_write_string("BOX UNLOCKED");
    LCD_command(LINE_TWO);
    LCD_write_string("WELCOME!");
    __delay_us(USER_READ);  // Delay to give user time to read the previous screen message
    LCD_init();
    LCD_write_string("# TO SET NEW PIN");  // Press "#" to enter a new pin for the lockbox
    LCD_command(LINE_TWO);
    LCD_write_string("* TO LOCK BOX"); // Press "*" to relock the box with the same pin used to unlock it
    key = "-1";  // Set key to default "not pressed" value
    while (strcmp(key, "-1") == 0)  // Wait until key press detected
```

```
{
    key = getkey();
}

 if (strcmp(key, "#") == 0)  // User sets a new password for the lockbox
 {
     LCD_init();
     LCD_write_string("ENTER NEW PIN:");
     LCD_command(LINE_TWO);
    __delay_us(USER_DELAY);
    memset(password,0,strlen(password));
    memset(new_password,0,strlen(new_password));
    counter = 0;
    while (counter < 4)
    {
        key = getkey();
        if (strcmp(key, "*") == 0)  // Clear LCD if "*" key is pressed
        {
            LCD_init();
            LCD_write_string("ENTER NEW PIN:");
            LCD_command(LINE_TWO);
            counter = 0;
            key = "-1";
            memset(password,0,strlen(password)); // Clear password string
        }
        if (strcmp(key, "-1") != 0)
        {
            __delay_us(USER_DELAY);  // Delay writing to LCD to give time for user input
            LCD_write_string(key);
            strcat(password, key);
            counter = counter + 1;  // Increment counter to check if entered pin is four or less digits
        }
    }
    __delay_us(USER_DELAY);
    LCD_init();
    LCD_write_string("NEW PIN SET");
    LCD_command(LINE_TWO);
    LCD_write_string("* TO LOCK BOX");
    counter = 1;
    while (counter == 1)
    {
        key = getkey();
        if (strcmp(key, "*") == 0)
        {
            counter = 0;
            P2->OUT = RED;
            state = KEY_PRESSED;
        }
    }
 }

 if (strcmp(key, "*") == 0)
 {
    counter = 0;
    memset(new_password,0,strlen(new_password));
    P2->OUT = RED;
    LCD_init();
    LCD_write_string("LOCKING BOX");
    __delay_us(USER_READ / 2);  // Entering dots slowly for dramatic effect
    LCD_write_string(".");
    __delay_us(USER_READ / 2);
    LCD_write_string(".");
```

```
                __delay_us(USER_READ / 2);
                LCD_write_string(".");
                __delay_us(USER_READ / 2);
                  state = KEY_PRESSED;
              }

              break;

          default:
              state = WAIT;  // Default for safety in case code fails
      }

  }
}
```

Keypad.h

```
// Keypad Header

#ifndef KEYPAD_H_
#define KEYPAD_H_

#define COLUMNS (BIT0 | BIT1 | BIT2 | BIT3)     // Set column bits
#define ROWS (BIT4 | BIT5 | BIT6 | BIT7)        // Set row bits
#define RGB_LED 0x07                            // Determines LED color of a specific button press
#define RED 0x01
#define GREEN 0x02

// Define Functions
void keypad_init(void);
const char* getkey(void);
void keypad_main(void);


#endif /* KEYPAD_H_ */
```

Keypad.c

```
// Keypad Source Code

#include "msp.h"
#include "Keypad.h"


void keypad_init(void){

    P5->SEL0 &= ~ROWS;      // Set rows and columns to GPIO
    P5->SEL1 &= ~ROWS;
    P4->SEL0 &= ~COLUMNS;
    P4->SEL1 &= ~COLUMNS;

    P4->DIR &= ~COLUMNS;    // Set P4.0 - 4.3 as inputs (columns)

    P5->DIR |= ROWS;        // Set P5.4 - 5.7 as outputs (rows)

    P4->REN |= COLUMNS;     // Enable pull down resistor on inputs (columns)
    P4->OUT &= ~COLUMNS;

    P5->REN |= ROWS;        // Set all rows = 1
    P5->OUT |= ROWS;

    return;
}


const char* getkey(void){
    char columns, columns_read, rows; // Initialize variables

    P5->OUT |= ROWS;
    columns = P4->IN & COLUMNS;       // Read columns to detect button press

    if (columns == 0x00){             // No button press detected
        char *key = "-1";
        return key;
    }

    rows = 0;
    P5->OUT &= ~ROWS;                 // Set all row outputs to 0

  while(rows < 4){
    P5->OUT = 0x10 << rows;           // Parse through each row
    __delay_cycles(25);               // Delay
    columns_read = P4->IN & COLUMNS;  // Reading columns

    if (columns_read != 0){           // Keypad press not detected
        break;
    }

    rows++;                           // Increment rows by 1 if key not found
  }

  if (rows == 0){                     // Returns value of key pressed
      if (columns == 0x01){
          char *key = "1";
          return key;
      }
      if (columns == 0x02){
```

```
        char *key = "2";
        return key;
    }
    if (columns == 0x04){
        char *key = "3";
        return key;
    }
    if (columns == 0x08){
        char *key = "A";
        return key;
    }
}

if (rows == 1){
    if (columns == 0x01){
        char *key = "4";
        return key;
    }
    if (columns == 0x02){
        char *key = "5";
        return key;
    }
    if (columns == 0x04){
        char *key = "6";
        return key;
    }
    if (columns == 0x08){
        char *key = "B";
        return key;
    }
}

if (rows == 2){
    if (columns == 0x01){
        char *key = "7";
        return key;
    }
    if (columns == 0x02){
        char *key = "8";
        return key;
    }
    if (columns == 0x04){
        char *key = "9";
        return key;
    }
    if (columns == 0x08){
        char *key = "C";
        return key;
    }
}

if (rows == 3){
    if (columns == 0x01){
        char *key = "*";
        return key;
    }
    if (columns == 0x02){
        char *key = "0";
        return key;
    }
    if (columns == 0x04){
        char *key = "#";
```

```
            return key;
        }
        if (columns == 0x08){
            char *key = "D";
            return key;
        }
    }

    if (rows > 4){
        char *key = "-1";      // Safety if a key press was missed
        return key;
    }

    char *key = "-1";          // Safety if a key press was missed
    return key;
}
```

LCD.h

```
// LCD Header

#ifndef LCD_H_
#define LCD_H_

#define CPU_FREQ 3000000    // Frequency of microcontroller
#define __delay_us(t_us) (__delay_cycles((((uint64_t)t_us)*CPU_FREQ) / 1000000))
#define RS_RW_ZERO 0x9F
#define RW_OUT 0xBF
#define RS_OUT 0x20
#define E_ZERO 0x7F
#define SHORT_DELAY_COMMAND 0x03
#define LCD_PARAM 0x28
#define CLEAR_DATA 0x0F
#define CLEAR_HOME 0x01
#define INC_CURSOR 0x06
#define CTRL_ZERO 0x1F
#define LINE_TWO 0xC0
#define USER_DELAY 300000
#define USER_READ 2000000

// Define Functions
void LCD_command(uint8_t);
void LCD_init(void);
void LCD_write_char(uint8_t);
void LCD_write_string(char[]);
void LCD_main(void);

#endif /* LCD_H_ */
```

LCD.c

```
// LCD Source Code

#include "msp.h"
#include "string.h"
#include "LCD.h"
#include "Keypad.h"

void LCD_command(uint8_t command)
{
    uint8_t high_nibble = command & ~CLEAR_DATA;    // Split command into high and low nibble
    uint8_t low_nibble = command << 4;
    P3->OUT &= RS_RW_ZERO;                          // Set RS (P3.5) and RW (P3.6) output = 0; AND w/ 1001_1111
    P4->OUT &= CLEAR_DATA;                          // Clear DB7 (P4.7) - DB4 (P4.4) before setting high nibble
    P4->OUT |= high_nibble;                         // Set DB7 (P4.7) - DB4 (P4.4) = high nibble; OR w/ high
    P3->OUT |= ~(E_ZERO);                           // Set E (P3.7) output = 1; OR w/ 1000_0000
    __delay_us(1);                                  // Delay by at least 140ns
    P3->OUT &= E_ZERO;                              // Set E (P3.7) output = 0; AND w/ 0111_1111
    __delay_us(2);                                  // Delay by at least 1200ns
    P4->OUT &= CLEAR_DATA;                          // Clear DB7 (P4.7) - DB4 (P4.4) before setting low nibble
    P4->OUT |= low_nibble;                          // Set DB7 (P4.7) - DB4 (P4.4) = low nibble; OR w/ low
    P3->OUT |= ~(E_ZERO);                           // Set E (P3.7) output = 1; OR w/ 1000_0000
    __delay_us(1);                                  // Delay by at least 140ns
    P3->OUT &= E_ZERO;                              // Set E (P3.7) output = 0; AND w/ 0111_1111
    if (command <= SHORT_DELAY_COMMAND){            // Delay by 1.53ms or 39us depending on command
        __delay_us(1530);
    }
    else{
        __delay_us(39);
    }
    return;
}


void LCD_init(void)
{
    P3->SEL0 &= CTRL_ZERO;                          // Set RS (P3.5) , RW (P3.6) , E (P3.7) to GPIO; AND w/ 0001_1111
    P3->SEL1 &= CTRL_ZERO;
    P4->SEL0 &= CLEAR_DATA;                         // Set DB7 (P4.7) - DB4 (P4.4) to GPIO; AND w/ 0000_1111
    P4->SEL1 &= CLEAR_DATA;
    P3->DIR |= ~(CTRL_ZERO);                        // Set RS (P3.5) , RW (P3.6) , E (P3.7) as outputs; OR w/ 1110_0000
    P4->DIR |= ~(CLEAR_DATA);                       // Set DB7 (P4.7) - DB4 (P4.4) as outputs
    __delay_us(40000);                              // Delay by at least 40ms
    P3->OUT &= RS_RW_ZERO;                          // Set RS (P3.5) and RW (P3.6) output = 0; AND w/ 1001_1111
    P4->OUT &= CLEAR_DATA;                          // Clear DB7 (P4.7) - DB4 (P4.4) before setting equal to 0x30
    P4->OUT |= 0x30;                                // Set DB7 (P4.7) - DB4 (P4.4) = 0x30; OR w/ 0011_0000
    P3->OUT &= ~(E_ZERO);                           // Set E (P3.7) output = 1; OR w/ 1000_0000
    __delay_us(1);                                  // Delay by at least 140ns
    P3->OUT &= E_ZERO;                              // Set E (P3.7) output = 0; AND w/ 0111_1111
    __delay_us(39);                                 // Delay by at least 39us
    LCD_command(LCD_PARAM);                         // 4-bit mode, 2 line, 5x8 font
    LCD_command(LCD_PARAM);                         // 4-bit mode, 2 line, 5x8 font
    LCD_command(CLEAR_DATA);                        // Display, cursor, and blink on
    LCD_command(CLEAR_HOME);                        // Clear home
    LCD_command(INC_CURSOR);                        // Increment cursor, no shift
    return;
}


void LCD_write_char(uint8_t letter)
```

```
{
    uint8_t high_nibble = letter & ~CLEAR_DATA;     // Split letter into high and low nibble
    uint8_t low_nibble = letter << 4;
    P3->OUT &= RW_OUT;                              // Set RW (P3.6) output = 0; AND w/ 1011_1111
    P3->OUT |= RS_OUT;                              // Set RS (P3.5) output = 1; OR w/ 0010_0000
    P4->OUT &= CLEAR_DATA;                          // Clear DB7 (P4.7) - DB4 (P4.4) before setting high nibble
    P4->OUT |= high_nibble;                         // Set DB7 (P4.7) - DB4 (P4.4) = high nibble; OR w/ high
    P3->OUT |= ~(E_ZERO);                           // Set E (P3.7) output = 1; OR w/ 1000_0000
    __delay_us(1);                                  // Delay by at least 140ns
    P3->OUT &= E_ZERO;                              // Set E (P3.7) output = 0; AND w/ 0111_1111
    __delay_us(2);                                  // Delay by at least 1200ns
    P4->OUT &= CLEAR_DATA;                          // Clear DB7 (P4.7) - DB4 (P4.4) before setting low nibble
    P4->OUT |= low_nibble;                          // Set DB7 (P4.7) - DB4 (P4.4) = low nibble; OR w/ low
    P3->OUT |= ~(E_ZERO);                           // Set E (P3.7) output = 1; OR w/ 1000_0000
    __delay_us(1);                                  // Delay by at least 140ns
    P3->OUT &= E_ZERO;                              // Set E (P3.7) output = 0; AND w/ 0111_1111
    __delay_us(43);                                 // Delay by at least 43us
    return;
}


void LCD_write_string(char string[])
{
    char i = 0;
    char length = strlen(string);                  // Get length of string
    for (i = 0; i < length; i++)                   // Use a for loop and LCD_write_char to print every letter of the
string
    {
        LCD_write_char(string[i]);
    }
}
```

External References

[1] *MSP432P4xx SimpleLink™ Microcontrollers Technical Reference Manual*, Texas Instruments, Mar. 2015 [Revised June 2019].

[2] Texas Instruments, “MSP432P401R, MSP432P401M SimpleLink™ Mixed-Signal Microcontrollers,” MSP432P401R datasheet, Mar. 2015 [Revised June 2019].

[3] Parallax, “4x4 Matrix Membrane Keypad (#27899),” Keypad datasheet, Dec. 2011.

[4] Orient Display, “SPECIFICATIONS FOR LCD MODULE AMC1602C ,” LCD datasheet, Aug. 1999 [Revised Mar. 2005] [Revised Again Dec. 2005].