Project 1: Hello World Electronic Lock

Cal Poly CPE EE 329

Spring 2018

April 23, 2018

Colleen Lau

Jonathan Richarte

https://youtu.be/9kewU8t7C_M

Purpose:

The purpose this project was to integrate the LCD and the keypad to create an electronic lock. The system is interfaced to an LCD that will display whether the user entered a correct or an incorrect key code on the keypad.

System Requirements:

1. Shall display LOCKED ENTER KEY on an LCD screen until the user enters the correct 4 digit key
2. Shall display the pressed keys of the keypad on the bottom row after KEY
3. Shall display HELLO WORLD if the key was corrected
4. Shall display the same LOCKED ENTER KEY screen and wait for a new key sequence if incorrect
5. Shall clear if the * key is pressed and display the LOCKED ENTER KEY screen until a new key sequence is entered

System Specification:

Component	Specifications	Value
MSP432	Variant	MSP-EXP432P401R
	Part Number	296-39653-ND
	Input Power	5V
	Vcc	5V
LCD	Variant	LCD NHD-0216HZ-FSW-FBW-33V3C File
	Part Number	113803-15-04-24
	Lines	2 lines
	Character Size	5 dots wide, 8 dots tall
	Character per line	16 characters
	Mode of operation	Nibble mode (4 bits)
Keypad	Variant	12 Button Solid Plastic
	Part Number	COM-08653
	Number of Keys	12
	Matrix (Columns * Rows)	3*4
	Output Type	Matrix

System Architecture:

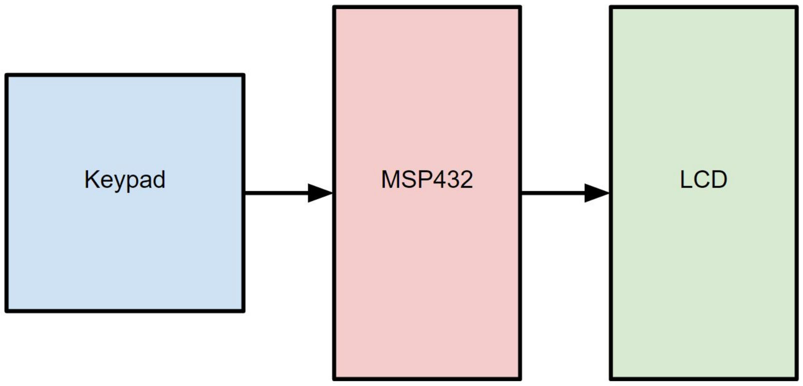

Figure 1: Overall system block diagram

Component Design:

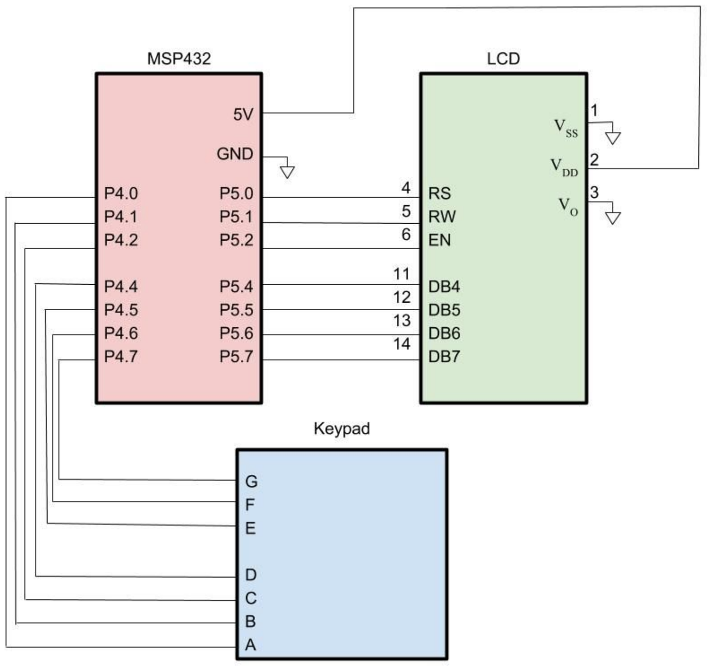

Figure 2: Schematic diagram of Keypad, MSP432, LCD

Bill of Materials:

Item #	Part Description	Supplier Name	Quantity	Price ($)	Extended Price ($)
1	MSP LaunchPad*	Texas Instruments	1	13.03	13.03
2	16x2 LCD Module*	Lumex Opto/Components In	1	6.01	6.01
3	12-key Keypad*	Adafruit Industries LLC	1	3.55	3.55
4	6" M/M Jumpers (Strip of 10)	Amazon	0.625	6.99	4.36875
5	6" M/F Jumpers (Strip of 10)	Amazon	0.625	-	-
* Part Numbers can be found in **Table 1: System Specifications**				Total	26.96

Table 2: Bill of Materials

System Integration:

Development Process:

In order to build the system, the LCD utilized Port 4 and the keypad utilized Port 5.
The LCD and keypad were first both tested separately with simple test code. The LCD was verified to print “Hello”, and the keypad was verified to light up an LED on the MSP432.
Once both systems were working successfully on their own, they were integrated using a simple program that used the LCD to display numbers pressed on the keypad.
The code for the combination lock program was developed independently of the hardware for unit testing. It was written in C and used the flowchart shown in Figure 2 as its design.
Once the hardware and software were both verified to be working, the process moved to integrating them. This involved constructing code to interpret keypad input in a format the software could understand, and moving software output onto the screen.

Significant Bugs and how we solved them:

Figuring what value the keypad returned was an issue. We solved by trial and error of lighting an LED when a specific value was read and testing every key to see what value that key corresponded to.

Project Demonstration:

https://youtu.be/9kewU8t7C_M

Conclusion:

Project 1's goal was to design an electronic lock using the LCD screen and the 12-key keypad. The system interfaced the MSP432 to take in the input from the keypad and display a message on the LCD indicating whether or not the user entered the correct code. To accomplish this, software needed to handle both keypad inputs and LCD outputs, as well as the underlying logic to create a locking system. This software had to interface timing constraints of the external hardware with the speed of the processor to create a cohesive product.

The project could be improved by adding an extra prompt for the user to know if their code was incorrect. It could also allow for reprogramming the lock code using only the hardware rather than changing software aspects.

The system could also be better designed for the real world by creating a custom PCB to house both the LCD and keypad without excess wiring. The system could be redesigned to run off a standard supply voltage to allow for battery power rather than USB power.

In addition, several improvements could be made to the software. The unlock key is currently stored in a standard character array, which is insecure and easy to access. A future implementation should contain security features to protect the password and prevent unauthorized access. The system also currently blocks waiting for input, and repeatedly probes the keypad for key presses. A future system could save power by using interrupts to read keypad actions and entering a low power mode between key presses.

MAIN.C

```c
#include "msp.h"
#include "LCD.h"
#include "keypad.h"
#include "delay.h"

/* sets all ports to GPIO, sets the DIR of all ports
 * to out, and all outputs are low. We do not want floating pins
 */
static void clear_pins() {
    P1->SEL0 = 0x00;
    P1->SEL1 = 0x00;
    P1->DIR = 0xFF;
    P1->OUT = 0x00;

    P2->SEL0 = 0x00;
    P2->SEL1 = 0x00;
    P2->DIR = 0xFF;
    P2->OUT = 0x00;

    P3->SEL0 = 0x00;
    P3->SEL1 = 0x00;
    P3->DIR = 0xFF;
    P3->OUT = 0x00;

    P4->SEL0 = 0x00;
    P4->SEL1 = 0x00;
    P4->DIR = 0xFF;
    P4->OUT = 0x00;

    P5->SEL0 = 0x00;
    P5->SEL1 = 0x00;
    P5->DIR = 0xFF;
    P5->OUT = 0x00;

    P6->SEL0 = 0x00;
    P6->SEL1 = 0x00;
    P6->DIR = 0xFF;
    P6->OUT = 0x00;

    P7->SEL0 = 0x00;
    P7->SEL1 = 0x00;
    P7->DIR = 0xFF;
    P7->OUT = 0x00;

    P8->SEL0 = 0x00;
    P8->SEL1 = 0x00;
    P8->DIR = 0xFF;
    P8->OUT = 0x00;

    P9->SEL0 = 0x00;
    P9->SEL1 = 0x00;
    P9->DIR = 0xFF;
    P9->OUT = 0x00;

    P10->SEL0 = 0x00;
    P10->SEL1 = 0x00;
    P10->DIR = 0xFF;
    P10->OUT = 0x00;
}

static void Write_Lock_Msg() {
    Clear_LCD();
    Write_string_LCD("LOCKED");
    New_Line();
    Write_string_LCD("ENTER KEY ");
}

static void Write_Success_Msg() {
    Clear_LCD();
    Write_string_LCD("Hello World");
}

// adds a flashy effect for different stages in the application
static void flash_LEDs(Byte value, int time, int loop) {
    int i;

    for (i = 0; i < loop; i++) {
        P2->OUT |= value;
        delay_ms(time, FREQ_3_MHz);
        P2->OUT &= ~value;
        delay_ms(time, FREQ_3_MHz);
    }
}

void main(void) {
    Byte button;
    int key = 0, key_len = 1000;

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    clear_pins(); // set all pins to output to avoid floating pins

    init_pins_LCD();
    init_pins_keypad();

    Startup_LCD();
    Clear_LCD();
    Home_LCD();

    Write_Lock_Msg();

    flash_LEDs(0x07, 500, 3);

    while(1) {
        button = get_keypad_button();

        if (button != 0) {          // a button was pressed
            if (button == '*') {   // reset screen
                Write_Lock_Msg();
                flash_LEDs(0x06, 100, 3);
                key = 0;            // reset our state variables
                key_len = 1000;
            }
            else if (key_len != 0) {    // still have room for more numbers
                Write_char_LCD(button);      // writes the number pressed
                delay_ms(200, FREQ_3_MHz);   // debounce
                // adds the number to the proper position
                key += (button - '0') * key_len;
                // moves our position in 'key' down by ten
                key_len /= 10;
                flash_LEDs(0x05, 25, 4);
            }
        }
        else if (key_len == 0) {      // get here once 4 numbers are pressed
            if (key == KEY) {
                Write_Success_Msg(); // check if it's the proper key
                flash_LEDs(0x02, 100, 5);
            }
            else {
                Write_Lock_Msg();
                flash_LEDs(0x01, 100, 5);
            }
            key = 0;    // reset our values, for the next time someone attempts
            key_len = 1000;
        }
    }
}
```

DELAY.H

```c
#ifndef DELAY_H
#define DELAY_H

#define FREQ_1_5_MHz 1500000
#define FREQ_3_MHz 3000000
#define FREQ_6_MHz 6000000
#define FREQ_12_MHz 12000000
#define FREQ_24_MHz 24000000
#define FREQ_48_MHz 48000000

void delay_ms(int ms, int freq);

void delay_us(int ms, int freq);

#endif
```

DELAY.C

```c
#include "msp.h"
#include "delay.h"

static void set_DCO(int freq) {
    CS->KEY = CS_KEY_VAL;

    CS->CTL0 = 0;

    if (freq == FREQ_1_5_MHz)
       CS->CTL0 = CS_CTL0_DCORSEL_0;
    else if (freq == FREQ_3_MHz)
       CS->CTL0 = CS_CTL0_DCORSEL_1;
    else if (freq == FREQ_6_MHz)
       CS->CTL0 = CS_CTL0_DCORSEL_2;
    else if (freq == FREQ_12_MHz)
       CS->CTL0 = CS_CTL0_DCORSEL_3;
    else if (freq == FREQ_24_MHz)
       CS->CTL0 = CS_CTL0_DCORSEL_4;
    else if (freq == FREQ_48_MHz) {
       // setting Vcore to level 1 for 48 MHz operation
       while(PCM->CTL1 & PCM_CTL1_PMR_BUSY);
       PCM->CTL0 = PCM_CTL0_KEY_VAL | PCM_CTL0_AMR_1;
       while(PCM->CTL1 & PCM_CTL1_PMR_BUSY);

       // configure flash wait-state to 1 for both banks 0 & 1
       FLCTL->BANK0_RDCTL = (FLCTL->BANK0_RDCTL &
           ~(FLCTL_BANK0_RDCTL_WAIT_MASK)) | FLCTL_BANK0_RDCTL_WAIT_1;
```

```
        FLCTL->BANK1_RDCTL = (FLCTL->BANK1_RDCTL &
                   ~(FLCTL_BANK1_RDCTL_WAIT_MASK)) | FLCTL_BANK1_RDCTL_WAIT_1;

        CS->CTL0 = CS_CTL0_DCORSEL_5;
     }

     CS->CTL1 &= ~(CS_CTL1_SELM_MASK | CS_CTL1_DIVM_MASK) | CS_CTL1_SELM_3;

     CS->KEY = 0;
}

// Delay milliseconds function
void delay_ms(int ms, int freq) {
   int i, j;

   set_DCO(freq);

   for (i = 0; i < ms; i++)
      for (j = .001 * freq / 10; j > 0; j--);   // delay 1 ms (approx)
}

// delay microseconds function
void delay_us(int us, int freq) {
   int i, j;

   set_DCO(freq);

   for (i = 0; i < us; i++)
      for (j = .00001 * freq / 10; j > 0; j--);    // delay 1 us (approx)
}
```

LCD.H

```
#ifndef LCD_H
#define LCD_H

void init_pins_LCD();

void Startup_LCD();

void Clear_LCD();

void Home_LCD();

void Write_char_LCD(char c);

void Write_string_LCD(char *str);
```

```c
void New_Line();

#endif
```

LCD.C

```c
#include "msp.h"
#include "delay.h"

typedef unsigned char Byte;

// set E(4.6) high, delay, then set it low
static void Send_Nybble() {
    P4->OUT |= BIT6;
    delay_ms(2, FREQ_3_MHz);
    P4->OUT &= ~BIT6;
}

// sets DB7-DB4 accordingly
static void Send_Upper_Four(Byte byte) {
    P4->OUT &= ~(0x0F);
    P4->OUT |= (byte >> 4);
}

static void Send_Command(Byte byte) {
    P4->OUT &= ~(BIT5 | BIT4);  // set both RS and R/W low

    // send upper nybble
    P4->OUT &= ~(0x0F);
    P4->OUT |= (byte >> 4);

    Send_Nybble();

    // send lower nybble
    P4->OUT &= ~(0x0F);
    P4->OUT |= (byte & 0x0F);

    Send_Nybble();
}

/* The following is assumed:
 * P4.7 is NO CONNECTION
 * P4.6 is E
 * P4.5 is RS
 * P4.4 is R/W
 * P4.3 is DB7
 * P4.2 is DB6
 * P4.1 is DB5
 * P4.0 is DB4
 * setup all of P4 as GPIO, even though we wont use 4.7
 */
void init_pins_LCD() {
    P4->SEL0 = 0x00;
    P4->SEL1= 0x00;
    P4->DIR = 0xFF; // All pins are output
}

void Startup_LCD() {
    P4->OUT &= ~(BIT5 | BIT4); // set RS and R/W low

    Send_Upper_Four(0x00);
    delay_ms(100, FREQ_3_MHz);

    Send_Upper_Four(0x30); // wake-up
    delay_ms(30, FREQ_3_MHz);

    Send_Nybble();
    delay_ms(10, FREQ_3_MHz);
    Send_Nybble();
    delay_ms(10, FREQ_3_MHz);
    Send_Nybble();
    delay_ms(10, FREQ_3_MHz);

    Send_Upper_Four(0x20);
    Send_Nybble();

    Send_Command(0x28);
    Send_Command(0x10);
    Send_Command(0x0F);
    Send_Command(0x06);
}

void Clear_LCD() {
    Send_Command(0x01);
    delay_ms(10, FREQ_3_MHz);
}

void Home_LCD() {
    Send_Command(0x02);
    delay_ms(10, FREQ_3_MHz);
}

void Write_char_LCD(char c) {
    P4->OUT |= BIT5;    // set RS high
    P4->OUT &= ~BIT4;   // set R/W low

    // send upper nybble
    P4->OUT &= ~(0x0F);
    P4->OUT |= (c >> 4);

    Send_Nybble();

    // send lower nybble
    P4->OUT &= ~(0x0F);
    P4->OUT |= (c & 0x0F);

    Send_Nybble();
}

// Writes a string to the LCD
void Write_string_LCD(char *str) {
    while(*str != '\0')
        Write_char_LCD(*(str++));
}

// Moves the cursor to the beginning of the second line
void New_Line() {
    Send_Command(0xC0);
    delay_ms(10, FREQ_3_MHz);
}
```

KEYPAD.H

```
#ifndef KEYPAD_H
#define KEYPAD_H

// password
#define KEY 3491

typedef unsigned char Byte;

void init_pins_keypad();

Byte get_keypad_button();

#endif
```

KEYPAD.C

```c
#include "msp.h"
#include "delay.h"

#define ONE 1
#define TWO 2
#define THREE 3
#define FOUR 5
#define FIVE 6
#define SIX 7
#define SEVEN 9
#define EIGHT 10
#define NINE 11
#define STAR 13
#define ZERO 14
#define HASHTAG 15

typedef unsigned char Byte;

/* uses all of P5 but 5.3
 * 5.0 to 5.2 are the Columns
 * 5.4 to 5.7 are the rows
 * 5.0 - C
 * 5.1 - A
 * 5.2 - E
 * 5.3 - NO CONNECTION
 * 5.4 - B
 * 5.5 - G
 * 5.6 - F
 * 5.7 - D
 */
void init_pins_keypad() {
    P5->SEL0 = 0x00;
    P5->SEL1 = 0x00;
    P5->DIR = 0x00; // set all pins as input
    P5->REN = 0x07; // only need internal resistors for 5.0 - 5.2
    P5->OUT = 0x07; // pull-up resistor for 5.0 - 5.2
}

/* values returned by get_keypad_button have a weird offset
 * this corrects that offset and will return the ascii value of 'byte'
 */
static Byte translate(Byte byte) {
    if (byte == ONE) return '1';
    else if (byte == TWO) return '2';
    else if (byte == THREE) return '3';
    else if (byte == FOUR) return '4';
    else if (byte == FIVE) return '5';
    else if (byte == SIX) return '6';
    else if (byte == SEVEN) return '7';
    else if (byte == EIGHT) return '8';
    else if (byte == NINE) return '9';
    else if (byte == ZERO) return '0';
    else if (byte == STAR) return '*';
    else if (byte == HASHTAG) return '#';

    return 0;
}

/* returns the position of the button that was pressed
 */
Byte get_keypad_button() {
    int row, col;
    char row_select[] = {0x10, 0x20, 0x40, 0x80}; /* one row is active */

    /* check to see any key pressed */
    P5->DIR |= 0xF0;            /* make all row pins output */
    P5->OUT &= ~0xF0;           /* drive all row pins low */
    delay_ms(1, FREQ_3_MHz);    /* wait for signals to settle */
    col = P5->IN & 0x07;        /* read all column pins */
    P5->OUT |= 0xF0;            /* drive all rows high before disable them */
    P5->DIR &= ~0xF0;           /* disable all row pins drive */

    if (col == 0x07)            /* if all columns are high */
        return 0;               /* no key pressed */

    /* If a key is pressed, it gets here to find out which key.
     * It activates one row at a time and read the input to see
     * which column is active. */
    for (row = 0; row < 4; row++) {
        P5->DIR &= ~0xF0;              /* disable all rows */
        P5->DIR |= row_select[row];    /* enable one row at a time */
        P5->OUT &= ~row_select[row];   /* drive the active row low */
        delay_ms(1, FREQ_3_MHz);       /* wait for signal to settle */
        col = P5->IN & 0x07;           /* read all columns */
        P5->OUT |= row_select[row];    /* drive the active row high */
        /* if one of the input is low, some key is pressed. */
        if (col != 0x07) break;
    }

    P5->OUT |= 0xF0;             /* drive all rows high before disable them */
    P5->DIR &= ~0xF0;            /* disable all rows */
    if (row == 4) return 0;       /* if we get here,no key is pressed */

    /* gets here when one of the rows has key pressed,
     * check which column it is
     */
    if (col == 0x06) return translate(row * 4 + 1);    /* key in column 0 */
    if (col == 0x05) return translate(row * 4 + 2);    /* key in column 1 */
    if (col == 0x03) return translate(row * 4 + 3);    /* key in column 2 */

    return 0; /* to be safe */
}
```