Digital Lock

Nathan Orloff

EE/CPE 329-13 Spring 2020

Project 1 – Digital Lock

23 April 2020

Dr. Malik

Video:

https://www.youtube.com/watch?v=ZVSiKlsX5Uc&feature=youtu.be

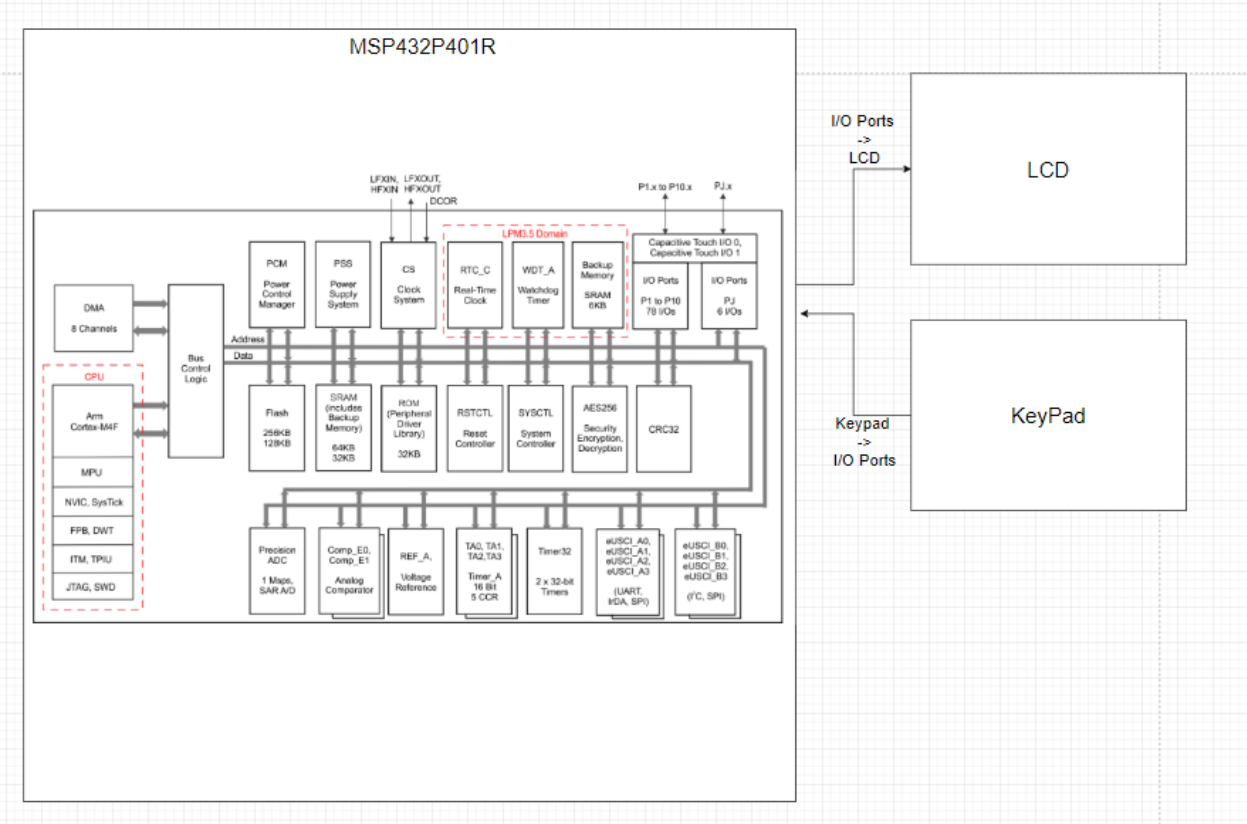
Behavior Description:

This device behaves as a locking system. During operation, the user inputs 4-digit codes onto a keypad, if the code the user inputs matches the code being used to lock the device, the device unlocks. If the user input does not match, then the device clears the user input and stays in its locked state. Each incorrect input from the user is tallied. If five incorrect codes are input the device will lock the user out preventing them from interacting with the device for five seconds. While the user is inputting codes onto the keypad, the user can clear their input. By pressing the star key (*) the device will clear the user input and enter its locked state, this does not add to the count of incorrect attempts. The user has the option to reset the code used to lock the device. By pressing the pound key (#) the user will be prompted to input the code currently being used to lock the device, before being able to input and confirm a new code. This new code will be used as the code to lock the device thereafter. During operation, the device will display which state it is in, along with user input, on a liquid crystal display.

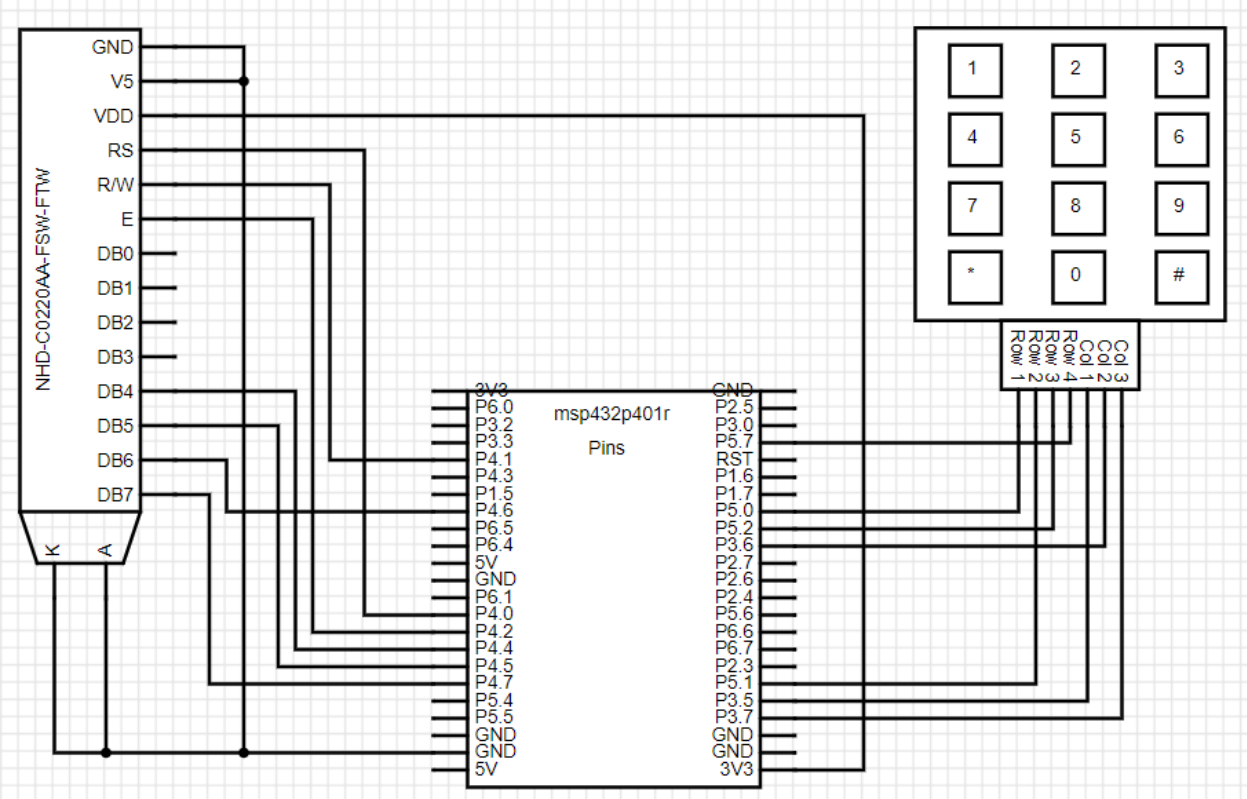
System Specifications:

Power Supply Voltage	5 v
Power Consumption	42 mW
Display Size	61.0 mm x 15.1 mm
Display Dot Size	0.45 mm x 0.65 mm
Clock Frequency	3 Mhz
Microcontroller Size	94 mm x 53 mm
Keypad Size	77mm x 68 mm

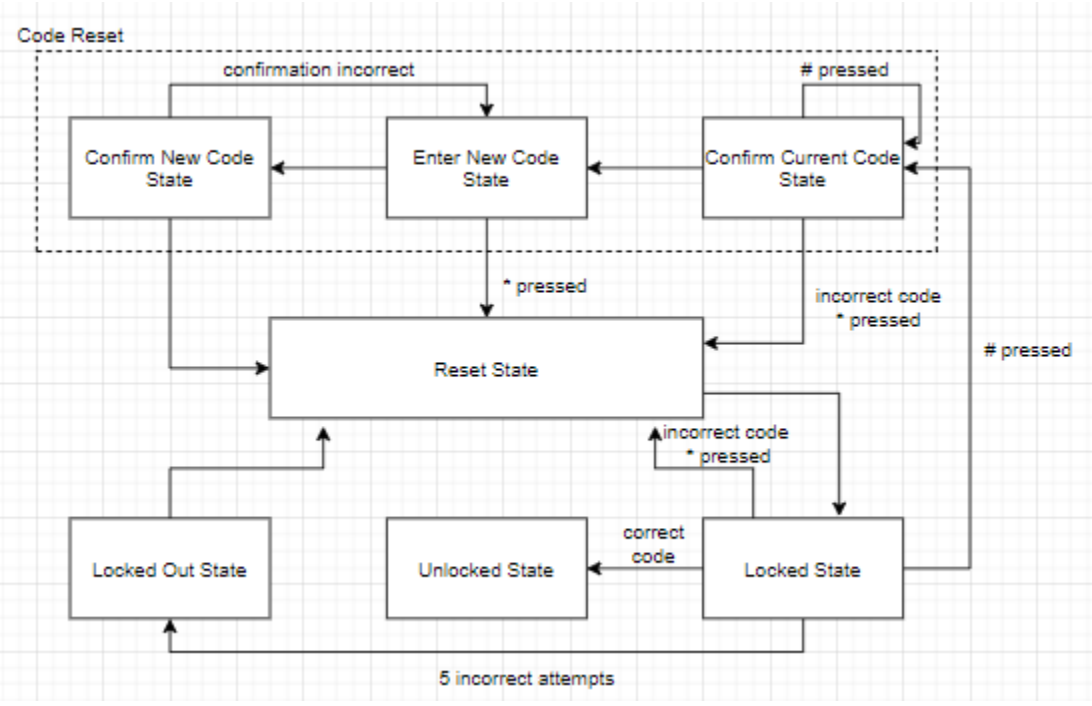
System Architecture:



System Schematic:



Software Architecture:



Bill of Materials:

Description	Item #	Part Number	Supplier Name	Quantity	Price (USD)	Extended Price (USD)
LAUNCHPAD MSP432P401R EVAL BRD	1	MSP-EXP432P401R	Digikey	1	23.59	23.59
LCD MOD 40DIG 20X2 TRANSFLCT WHT	2	NHD-C0220AA-FSW-FTW	Digikey	1	10.85	10.85
SWITCH KEYPAD 12 KEY NON-ILLUM	3	419	Digikey	1	3.95	3.95
JUMPER WIRE F/F 6" 20PCS	4	PRT-12796	Digikey	1	1.95	1.95
					Total Price (USD): 40.34	

Ethics Implications:

This project carries a few ethical implications with it. From an environmental perspective this product does use electrical energy. Electrical energy is created in a number of ways some of which can affect global warming. This device uses a small amount of electrical energy and does not have a significant impact on the environment. From a social perspective, this device has the ability to be used to lock away certain items. This, depending on how it is being used, can benefit or harm people. This gives a safe way to store important documents, but can also be used to store illegal goods.

```c
/*Nathan Orloff
 * EE/CPE 329
 * This program runs a digital lock system. the user types in 4-digit codes
 * on
 * a keypad and tries to unlock the lock by matching the code being used to
 * lock
 * the device. The state of the device and the input from the user are
 * displayed
 * on the LCD display in nibble mode*/
#include "keypad.h"
#include "LCD.h"
#include "msp.h"
#include <stdint.h>
#include <stdio.h>

int check(char a[], char b[]);
void set_code(void);



/*creates states to be used in a FSM*/
enum states{reset, locked, unlocked, confirm_old, type_new, confirm_new,
locked_out};



char code[] = {'1', '1', '1', '1'}; //holds current code for digital lock,
                                    //initial code is "1111"
char input[4];  //holds the user input from the keypad
char new_code[4];   //holds the new code when in process of changing the code



void main(void)
{
    int PS; //present state
    int NS = reset; //next state, initially equal to reset state

    int incorrect_count = 0;    //count of incorrect code attempts
    int end = 0;        //once in unlocked state turns to 1 ends the program
    int wait = 0;

    LCD_init();         //initialize the LCD
    Clear_LCD();        //Clear the Display
    Home_LCD();         //set cursor at first position of first line
    keypad_init();      // setup gpio pins for keypad

    while(end == 0){    //while not unlocked run program
        PS = NS;    //update the present state

        switch(PS){
            case reset: //reset state
                Clear_LCD();        //Clear the Display
                Home_LCD();
                NS = locked;    //reset state always goes to locked state
                break;
            case locked:    //locked state
```

```
        Write_string_LCD("Locked");
        Line_two_LCD();
        Write_string_LCD("Enter Key ");
        int in = 0;      //variable used to count user input
        uint8_t ikey;

        int star_detected = 0;  //if * is detected turns to 1
        int pound_detected = 0; //if # is detected turns to 1
        /* while statement gets the user input 4 times as long as the
         * input is not
         * * or # it will write the input to the display and to the
         * input matrix
         * does not continue if * or # are the input*/
        while(in < 4 && star_detected == 0 && pound_detected == 0){
            ikey = keypad_getkey(); //gets user input as integer
            char ckey = int_to_char(ikey);  // as a char
            if(ikey != 255 && wait == 0){ /*write value to display*/
                if(ckey == '*'){
                    star_detected = 1;
                }
                if(ckey == '#'){
                    pound_detected = 1;
                }
                Write_char_LCD(ckey);
                input[in] = ckey;   //assign user input
                wait = 1;   // input to only be displayed once
                in++;
            }else if (ikey == 255){ /*key must be released */
                wait = 0;
            }
        }

        if(star_detected == 0 && pound_detected == 0){
            int correct_code = check(input, code);
            if(correct_code == 0){NS = unlocked;}
            //if code is correct next state is unlocked
            else{
                incorrect_count++;  // incorrect count increased
                NS = reset;
            }
        }

        if(star_detected == 1){ //if a * was input, next state reset
            NS = reset;
        }
        if(pound_detected == 1){    // next state confirm old
            NS = confirm_old;
        }
        if(incorrect_count == 5){
       //if 5 incorrect attempts were input next state locked out
            NS = locked_out;
        }
        break;
    case unlocked:  //unlocked state
        Clear_LCD();    //reset display
        Home_LCD();
        Write_string_LCD("Unlocked");
```

```
        end = 1;    //will end the program
        break;
    case confirm_old:   //confirm old code state
        Clear_LCD();    //reset display
        Home_LCD();
        Write_string_LCD("Confirm Old Code");
        Line_two_LCD();
        Write_string_LCD("Enter Key ");
        in = 0;


        star_detected = 0;
        pound_detected = 0;
        /* while statement gets the user input 4 times as long as the
         * input is not
         * * or # it will write the input to the display and to the
         * input matrix
         * does not continue if * or # are the input*/
        while(in < 4 && star_detected == 0 && pound_detected == 0){
            ikey = keypad_getkey(); //gets user input as an integer
            char ckey = int_to_char(ikey);  // as a character
            if(ikey != 255 && wait == 0){ /*write value to display*/
                if(ckey == '*'){
                    star_detected = 1;
                }
                if(ckey == '#'){
                    pound_detected = 1;
                }
                Write_char_LCD(ckey);
                input[in] = ckey;  //assign user input
                wait = 1;   // input to only be displayed once
                in++;
             }else if (ikey == 255){ /*key must be released */
                    wait = 0;
                }
         }

         if(star_detected == 0 && pound_detected == 0){
             int correct_code = check(input, code);
             if(correct_code == 0){NS = type_new;}
            //if input matches the code, next state is type new
             else{
                 NS = reset;    //if not a match, next state is reset
             }
         }

         if(star_detected == 1){    // next state is reset
             NS = reset;
         }
         if(pound_detected == 1){   // state is confirm old
             NS = confirm_old;
         }

        break;
    case type_new:  //type new code state
        Clear_LCD();    //reset display
        Home_LCD();
```

```
        Write_string_LCD("Enter New Code");
        Line_two_LCD();
        Write_string_LCD("Enter Key ");
        in = 0;


        star_detected = 0;
        /* while statement gets the user input 4 times as long as the
         * input is not
         * * it will write the input to the display and to the
         * new_code matrix
         * does not continue if * is the input*/
        while(in < 4 && star_detected == 0){
            ikey = keypad_getkey(); //gets user input as integer
            char ckey = int_to_char(ikey);  //as a character
            if(ikey != 255 && wait == 0){ /* write to display once*/
                if(ckey == '*'){
                    star_detected = 1;
                }

                if(ckey != '#'){    // ignores #
                    Write_char_LCD(ckey);
                    new_code[in] = ckey;    //stores user input
                    wait = 1;   // only be displayed once
                    in++;
                }
            }else if (ikey == 255){ /*key must be released
                    wait = 0;
            }
        }

        NS = confirm_new;   //next state is confirm new unless *

        if(star_detected == 1){ //next state is reset if *
            NS = reset;
        }

        break;
    case confirm_new:   //confirm the new code state
        Clear_LCD();    //reset display
        Home_LCD();
        Write_string_LCD("Confirm New Code");
        Line_two_LCD();
        Write_string_LCD("Enter Key ");
        in = 0;
        /* while statement gets the user input 4 times
         * write the input to the display and to the input matrix*/
        while(in < 4 && star_detected == 0 && pound_detected == 0){
            ikey = keypad_getkey(); //get user input as an integer
            char ckey = int_to_char(ikey);  // as a character
            if(ikey != 255 && wait == 0){ /*value to display once*/

                if(ckey != '*' && ckey != '#'){ //ignore * and #
                    Write_char_LCD(ckey);
                    input[in] = ckey;   //store character in
                    wait = 1;   // input to only be displayed once
                    in++;
```

```
                    }
                }else if (ikey == 255){ /*key must be released
                        wait = 0;
                }
            }


            /*if the input matches the new_code previously input
             * the new_code is set as the code for the digital lock
             * the next state is set to reset
             * if the input does not match the new_code then the
             * next state is set the type_new*/
            if(check(input, new_code) == 0){
                set_code();
                NS = reset;
            }else{
                NS = type_new;
            }

            break;
        case locked_out:    //locked out state
            Clear_LCD();    //resets display
            Home_LCD();
            Write_string_LCD("Locked Out");
            incorrect_count = 0;    //resets the user attempts to zero
            delayMs(5000);      //locks user out for 5 seconds
            NS = reset;     //next state is the reset state
            break;
    }

  }
}

/* checks to see if two char matrices of length 4 contain
 * the same characters in the same order. If all characters
 * are the same and in the same order, the function returns 0
 * if they are unequal then the function returns 1
 **/
int check(char a[], char b[]){
    int not_equal = 0;
    int i;
    for(i = 0; i < 4; i++){
        if(a[i] != b[i]){
            not_equal = 1;
        }
    }
    return not_equal;
}

/* The char values to unlock the digital lock is set to the
 * char values the user input*/
void set_code(void){
    int i;
    for(i = 0; i < 4; i++){
        code[i] = input[i];
    }
}
```

```c
/*Nathan Orloff
 * keypad.h
 * Header file that declares all of the
 * functions of the associated C file
 * */


#include "msp.h"
#include <stdint.h>
#include <stdio.h>

#define COL1  BIT5
#define COL2  BIT6
#define COL3  BIT7
#define ROW1  BIT0
#define ROW2  BIT1
#define ROW3  BIT2
#define ROW4  BIT7

uint8_t keypad_getkey(void);
void keypad_init(void);
char int_to_char(int a);
```

```c
/*Nathan Orloff
 * keypad.c
 * C file associated with its header file
 * creates functions for performing operations with the keypad
 * includes initializing, detecting a key pressed and getting the value
 * of the key that was pressed
 *  */

#include "msp.h"
#include <stdint.h>
#include <stdio.h>

#define COL1  BIT5
#define COL2  BIT6
#define COL3  BIT7
#define ROW1  BIT0
#define ROW2  BIT1
#define ROW3  BIT2
#define ROW4  BIT7


/* this function initializes Ports 2 and 3 that is connected to the keypad.
 * All pins are configured as GPIO input pin. The row pins have
 * the pull-down resistors enabled.
 */
void keypad_init(void) {
    P3->DIR = 0;         // make all pins an input
    P5->DIR = 0;     //make pins as inputs
    P5->REN |= (ROW1 | ROW2 | ROW3 | ROW4);  // enable resistor for row pins
    P5->OUT &= ~(ROW1 | ROW2 | ROW3 | ROW4); // make row pins pull-down
}


/*
 * This is a non-blocking function to read the keypad.
 * If a key is pressed, it returns that key value 0-9. * is 10, # is 12
 * If no key is pressed, it returns 0xFF
 * Port 2.4 - 2.7 are used as inputs and connected to the rows. Pull-down
 * resistors are enabled so when no key is pressed, these pins are pulled low
 *
 * The Port 3.5 - 3.7 are used as outputs that drives the keypad columns.
 * First all columns are driven high and the input pins are read. If no key
 * is
 * pressed, they will read zero because of the pull-down resistors. If no key
 * is pressed, return 0xFF. If the value is non-zero, determine which key is
 * being pressed.
 * To determine which key is being pressed, the program proceeds to drive one
 * column high at a time and read the input pins (rows). Knowing which row is
 * high and which column is active, the program can decide which key is
 * pressed
 *
 */
uint8_t keypad_getkey(void) {
    uint8_t row, col, key;

    /* check to see any key pressed */
    P3->DIR |= (COL1 | COL2 | COL3);  // make the column pins outputs
```

```
    P3->OUT |= (COL1 | COL2 | COL3);  // drive all column pins high
    _delay_cycles(25);                // wait for signals to settle

    row = P5->IN & (ROW1 | ROW2 | ROW3 | ROW4);   // read all row pins

    if (row == 0)           // if all rows are low, no key pressed
        return 0xFF;

    /* If a key is pressed, it gets here to find out which key.
     * It activates one column at a time and reads the input to see
     * which row is active. */

    for (col = 0; col < 3; col++) {
        // zero out bits 6-4
        P3->OUT &= ~(COL1 | COL2 | COL3);

        // shift a 1 into the correct column depending on which to turn on
        P3->OUT |= (COL1 << col);
        _delay_cycles(25);            // wait for signals to settle

        row = P5->IN & (ROW1 | ROW2 | ROW3 | ROW4); // mask only the row pins

        if (row != 0) break;      // if the input is non-zero, key detected
    }

    P3->OUT &= ~(COL1 | COL2 | COL3);   // drive all columns low
    P3->DIR &= ~(COL1 | COL2 | COL3);   // disable the column outputs

    if (col == 3)   return 0xFF; // if we get here, no key was detected

    // rows are read in binary, so powers of 2 (1,2,4,8)
    if (row == 4) row = 3;
    if (row == 8) row = 4;

    /*********************************************************************
     * IF MULTIPLE KEYS IN A COLUMN ARE PRESSED THIS WILL BE INCORRECT *
     *********************************************************************/

    // calculate the key value based on the row and columns where detected
    if (col == 0) key = row*3 - 2;
    if (col == 1) key = row*3 - 1;
    if (col == 2) key = row*3;

    if (key == 127)  key = 0; // fix for 0 key



    return key;
}

/* takes an integer and returns that integer as a char
 * special cases cause it to return * or # depending on the
 * input integer*/
char int_to_char(int a){
    char disp[1];
    if (a == 126){ // if key presses is * or #, assign those chars to disp[0]
        disp[0] = '*';
```

```
    }else if(a == 128){
        disp[0] = '#';
    }else{
        sprintf(disp, "%d", a);  //int to char conversion
    }
    return disp[0];
}
```

```c
/*Nathan Orloff
 * LCD.h
 * Header file that declares all of the
 * functions of the associated C file
 * */

#include "msp.h"
#define RS 1     /* P4.0 mask */
#define RW 2     /* P4.1 mask */
#define EN 4     /* P4.2 mask */

void LCD_nibble_write(unsigned char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);
void Clear_LCD();
void Home_LCD();
void Line_two_LCD();
void Write_char_LCD(unsigned char data);
void delayMs(int n);
void Write_string_LCD(unsigned char* data);
```

```c
/*Nathan Orloff
 * LCD.c
 * C file associated with its header file
 * creates functions for performing operations on the LCD
 * includes initializing, clearing, setting the cursor, writing to the LCD
 *  */

#include "msp.h"


#define RS 1     /* P4.0 mask */
#define RW 2     /* P4.1 mask */
#define EN 4     /* P4.2 mask */

void delayMs(int n) {
    int i, j;

    for (j = 0; j < n; j++)
        for (i = 750; i > 0; i--);      /* Delay */
}

void LCD_init(void) {   /* initializes the LCD */
    P4->DIR = 0xFF;     /* make P4 pins output for data and controls */
    delayMs(30);                /* initialization sequence */
    LCD_nibble_write(0x30, 0);
    delayMs(10);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x20, 0);  /* use 4-bit data mode */
    delayMs(1);

    LCD_command(0x28);      /* set 4-bit data, 2-line, 5x7 font */
    LCD_command(0x06);      /* move cursor right after each char */
    LCD_command(0x01);      /* clear screen, move cursor to home */
    LCD_command(0x0F);      /* turn on display, cursor blinking */
}

/* With 4-bit mode, each command or data is sent twice with upper
 * nibble first then lower nibble.
 */
void LCD_nibble_write(unsigned char data, unsigned char control) {
    data &= 0xF0;           /* clear lower nibble for control */
    control &= 0x0F;        /* clear upper nibble for data */
    P4->OUT = data | control;       /* RS = 0, R/W = 0 */
    P4->OUT = data | control | EN;  /* pulse E */
    delayMs(0);
    P4->OUT = data;                 /* clear E */
    P4->OUT = 0;
}

void LCD_command(unsigned char command) {/* executes command for the LCD */
    LCD_nibble_write(command & 0xF0, 0);    /* upper nibble first */
    LCD_nibble_write(command << 4, 0);      /* then lower nibble */

    if (command < 4)
```

```
        delayMs(4);          /* commands 1 and 2 need up to 1.64ms */
    else
        delayMs(1);          /* all others 40 us */
}

void LCD_data(unsigned char data) {     /* writes data to LCD */
    LCD_nibble_write(data & 0xF0, RS);     /* upper nibble first */
    LCD_nibble_write(data << 4, RS);       /* then lower nibble  */

    delayMs(1);
}

void Clear_LCD(){   /* clear the display */
    LCD_nibble_write(1 & 0xF0, 0);     /* upper nibble first */
    LCD_nibble_write(1 << 4, 0);       /* then lower nibble */

    delayMs(4);          /* commands 1 and 2 need up to 1.64ms */

}

void Home_LCD(){    /* move the cursor to the top left of the LCD */
    LCD_nibble_write(0x80 & 0xF0, 0);     /* upper nibble first */
    LCD_nibble_write(0x80 << 4, 0);       /* then lower nibble */

    delayMs(1);          /* all others 40 us */
}

void Write_char_LCD(unsigned char data){     /* write a character to the LCD
*/
    LCD_nibble_write(data & 0xF0, RS);     /* upper nibble first */
    LCD_nibble_write(data << 4, RS);       /* then lower nibble  */

    delayMs(1);
}

void Line_two_LCD(){        /* move the cursor to the bottom left of the LCD
*/
    LCD_nibble_write(0xC0 & 0xF0, 0);     /* upper nibble first */
    LCD_nibble_write(0xC0 << 4, 0);       /* then lower nibble */

    delayMs(1);          /* all others 40 us */
}

void Write_string_LCD(unsigned char* letter){/* writes a string to the LCD */
    int i = 0;
    while(letter[i]!= '\0'){  /* while not at the end of the string */
        LCD_nibble_write(letter[i] & 0xF0, RS);     /* upper nibble first */
        LCD_nibble_write(letter[i] << 4, RS);       /* then lower nibble  */
        i++;
        delayMs(1);
    }

}
```

References:

- LaunchPad User's Guide (LUG)
  http://www.ti.com/lit/ug/slau597f/slau597f.pdf?ts=1587852043084
- MSP432P401 Datasheet (MDS)
  http://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=http%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Fmsp432p401r
- MSP432 Technical Reference Manual (TRM)
  https://www.ti.com/lit/ug/slau356i/slau356i.pdf?ts=1587852104208