Digital Password Lock

Mark Nakamae

Cal Poly SLO

March 2020

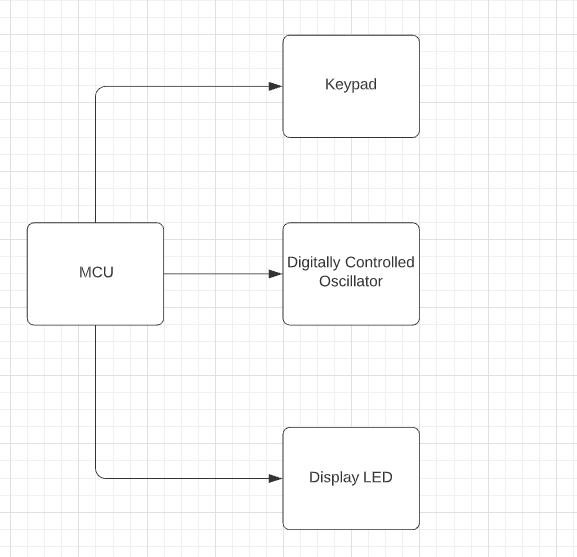
**Behavior Description**

This lock is a keypad lock that starts off in the locked stage. Once the correct 4 digit pin has been entered the lock unlocks. Once in the unlocked stage, the user can press the # button to reset their pin and will be prompted to insert a new 4 digit pin. After the new pin has been inserted, the lock will lock. In the unlocked stage pressing any button besides the ‘#’ will return the user to the locked stage with the same pin. If the user inserts the wrong pin 3 times in a row they will get put in timeout for 10 seconds before being brought back to the locked stage to attempt guesses at the pin. Lastly in the locked stage if the user presses the ‘\*’ it will clear their input and will not count as an attempt.

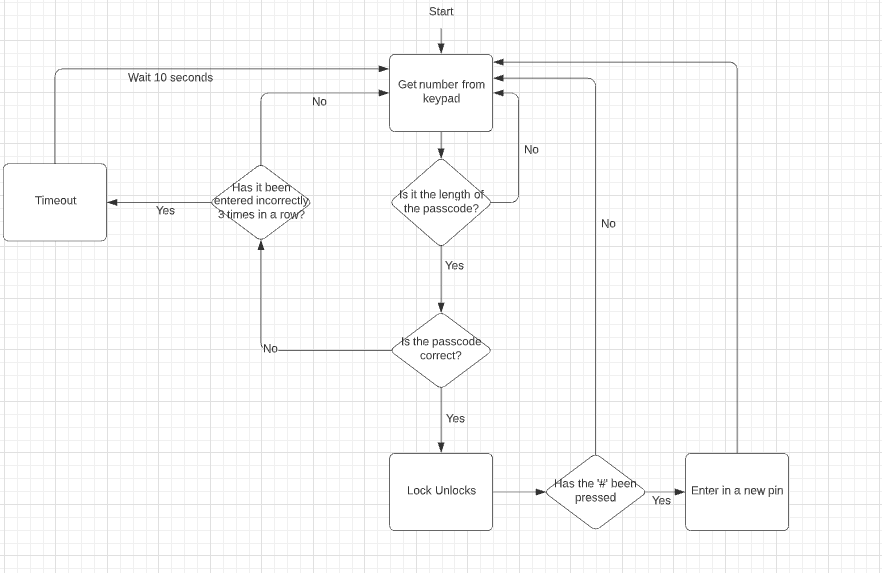
**System Specification**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Power Supply Voltage** | **Clock Frequency** | **Size of keypad**  **(cm)** | **Size of MSP432**  **(cm)** | **Size of LCD screen**  **(cm)** | **Response time of keypad** |
| 3.3 V | 3 MHz | 6.6 x 7.6 | 5.7 x 9.2 | 2.5 x 7.5 | 1 second |

**System Architecture**

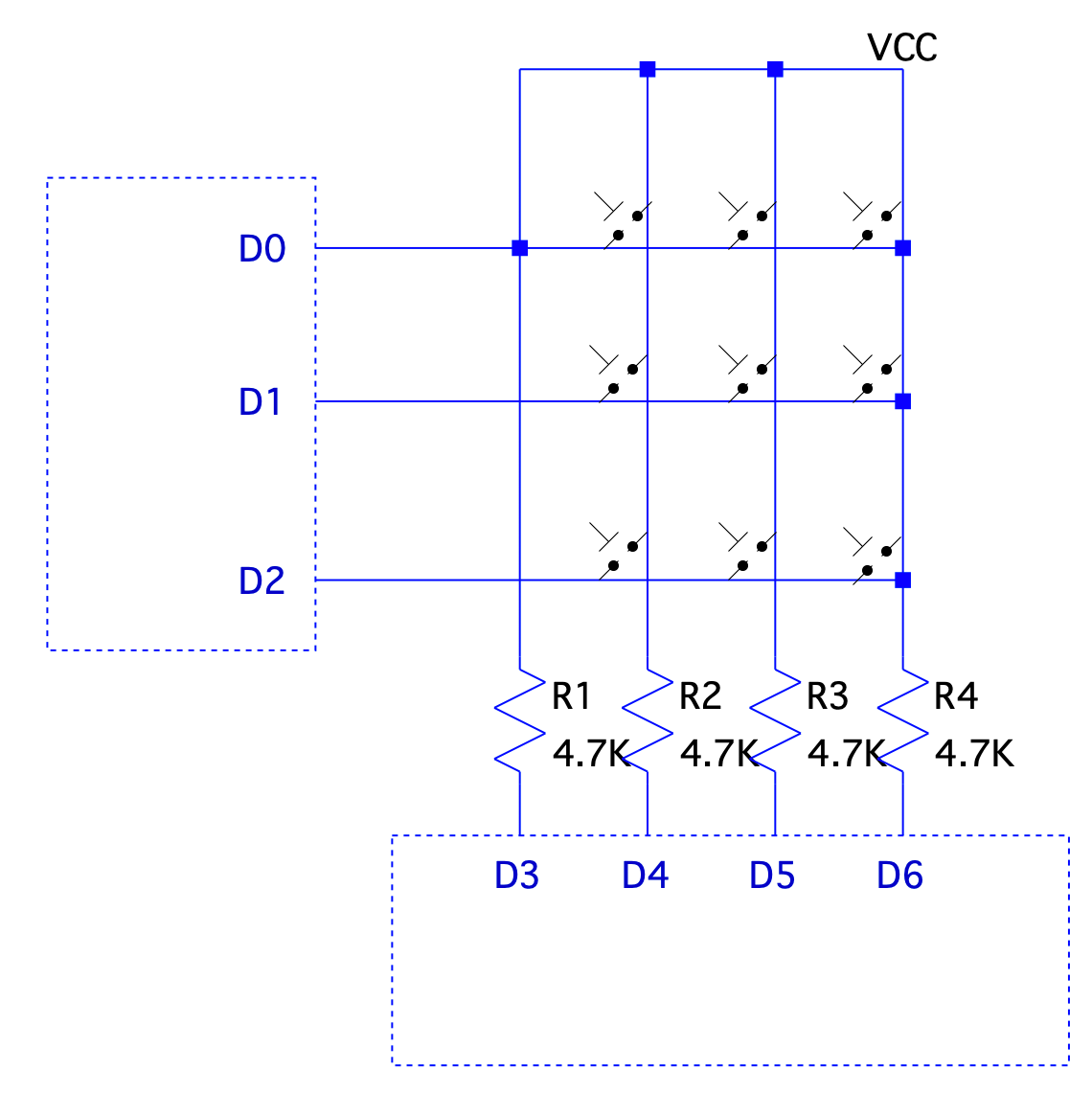
****

**Figure 1, Black Box Diagram**: This is a high-level schematic of all the components involved in the keypad lock device.

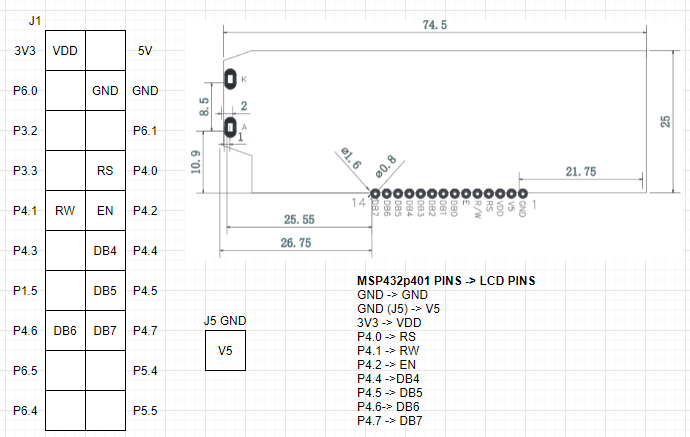


**Figure 2, Flow Chart**: This is a flow chart of how the program functions.

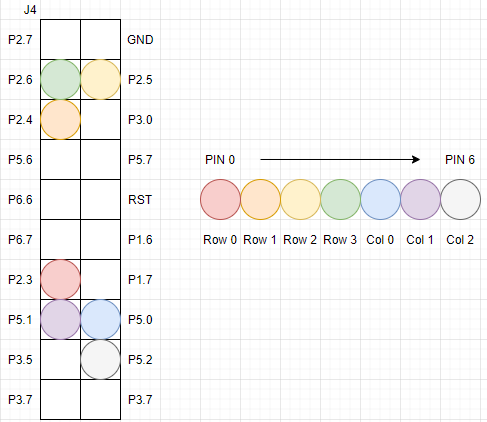
**System Schematic**

****

**Figure 3, Low level schematic**: This shows the interconnections of the device.

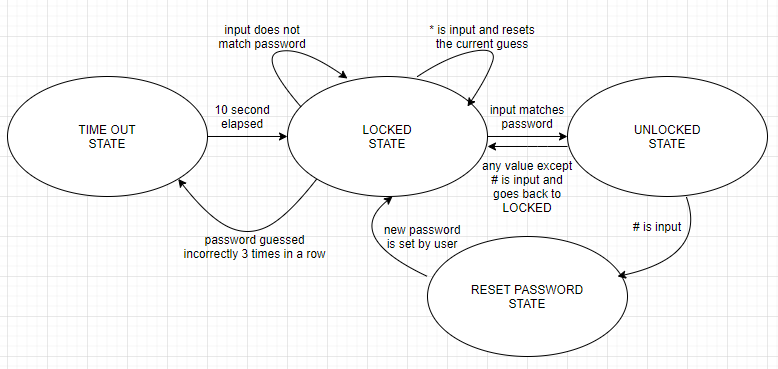


**Figure 4, LCD pinning**: This diagram labels the pins connecting the LCD to the MSP432P401R board. Note that J1 and J3 on the board is used for all GPIO related to the LCD.



**Figure 5, Keypad Pinnings**: This diagram labels the pins connecting the keypad to the MSP432P401R board. Pin 0 on the keypad corresponds to row 0, and pin 6 corresponds to column 2. Note that J2 and J4 on the board is used for all GPIO related to the keypad.

**Software Architecture**



**Figure 6, FSM Logic Flowchart:** This diagram shows the finite state machine and its process flow. Each condition for a state to move onto the next state is listed next to each arrow. This FSM is initially set to the locked state.

**Bill of Materials**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ITEM NUMBER** | **PART NUMBER** | **ITEM** | **SUPPLIER** | **QUANTITY** | **PRICE EACH** | **PRICE EXTENDED** |
| 1 | 296-39653-ND | LaunchPad MSP432P401R Board | Texas Instruments | 1 | $23.59 | $23.59 |
| 2 | 1568-1511-ND | Jumper Wire M/F 6” 20 Pieces | SparkFun Electronics | 1 | $1.95 | $1.95 |
| 3 | 1528-1136-ND | Switch Keypad 12 Key Non-ILLUM | Adafruit Industries LLC | 1 | $3.95 | $3.95 |
| 4 | NHD-C0220AA-FSW-FTW-ND | NHD-C0220AA-FSW-FTW LCD Display | Newhaven Display Intl | 1 | $8.82 | $8.82 |
| 5 | EL-CP-003 | 830 Point Solderless Breadboard 3PCS | ELEGOO | 1 | $8.99 | $8.99 |
| TOTAL | | | | | | **$47.30** |

**Ethical Implications**

**Confidentiality & Privacy**: A digital lock like this one can be used to keep important data private and confidential. Without a security measure such as this one, people could access the contents of the machine that the lock is on. This is a breach of both confidentiality and privacy. Thus pincodes are helpful in stopping those who want to steal out information. The Fourth Amendment of the Constitution states that people have a right to be secure in homes and against unreasonable search and seizure. A security measure like this is just one option in keeping your data safe and secure. (Although I wouldn’t want this exact lock as it is a little lacking in the safe and secure front.) All in all, locks serve an important role in protecting our rights.

**Societal**: Like anything else, this lock is far from perfect. Although it keeps people from accessing “sensitive information”, its implementation is more meaningful serving the role as a deterrent for theft. If someone really wanted to bypass this security, they could easily accomplish this feat given enough time. However, these locks are mostly implemented to keep honest people honest, while giving people with poor intentions another obstacle to overcome. Thus, these locks serve as a test of character.

**Political:** Locks should be designed with the sole purpose of securing the user’s goods or information regardless of race, age, or any other condition. That means for the engineers designing these locks, they should design products that are not biased towards the user. Besides locks being secured, they should not have a backdoor that allows someone with power or money to infiltrate someone’s private information. Also countries should have locks that protect sensitive information from other countries or else this could have negative implications such as swaying elections in a candidates favor, new technologies a country is developing for its military, or new economic plans. Locks are necessary to carry out their main function as well as not have flaws to compromise private information.

**Appendices**

//this is the main

#include "msp.h"

#include "delay.h"

#include "lcd.h"

#include "keypad.h"

#define LOCKED &FSM[0]

#define UNLOCKED &FSM[1]

#define TIMEOUT &FSM[2]

#define RESET &FSM[3]

#define PASSWORD "1234"

#define SECOND 1000

#define PASSLEN 4

struct State

{

struct State \*Next[3];

};

struct State FSM[4] =

{

{{LOCKED, UNLOCKED, TIMEOUT}}, //LOCKED state

{{UNLOCKED, LOCKED, RESET}}, //UNLOCKED state

{{TIMEOUT, LOCKED,LOCKED}}, //TIMEOUT state

{{RESET,LOCKED, LOCKED}} //RESET state

};

//this function prints the two strings on the two different lines

void print\_screen(char line1[], char line2[])

{

clear\_LCD();

write\_string\_LCD(line1);

LCD\_command(0xC0);

write\_string\_LCD(line2);

}

//compares the two strings and returns 0 if they are equal, 42 else

int comp(char str1[], char str2[])

{

int i;

for (i=0; i<PASSLEN; i++)

{

if (str1[i] != str2[i])

{

return 42;

}

}

return 0;

}

/\*\*

\* main.c

\* this is the main of the function and it just consists of a finite state machine

\*/

void main(void)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

char lock1[]= "LOCKED";

char lock2[]= "ENTER KEY ";

char time1[]= "STOP YOU'VE BROKEN";

char time2[]= "THE LAW ";

char res1[]= "RESET PASSWORD";

char res2[]= "ENTER PASS: ";

char ulock1[]= "UNLOCKED";

char ulock2[]= "# to change PASSWORD ";

char nums[] = "0123456789\*0#";

char guess[PASSLEN];

char key[PASSLEN]= PASSWORD;

char next = 0, flag = 0, tries = 0;;

char i = 0;

struct State \*ptr= LOCKED ; //state pointer

LCD\_init();

keypad\_Init();

set\_DCO(FREQ\_3MHz);

delay\_ms(SECOND);

while (1)

{

next = 0;

if (ptr == LOCKED)

{

if (flag == 0)

{

print\_screen(lock1,lock2);

flag++;

}

if(i == PASSLEN) //check guess length against password length

{

i = 0;

flag = 0;

if(!comp(guess, key))

{

tries = 0;

next = 1;

}

else //wrong guess

{

tries++;

}

if (tries == 3)

{

next = 2;

}

}

char in = keypad\_Getkey();

if (in != 0xFF)

{

if (nums[in] == '\*') // clear the screen

{

print\_screen(lock1,lock2);

i = 0;

}

else if (nums[in] !='#' ) // write to the screen

{

write\_char\_LCD(nums[in]);

guess[i] = nums[in];

i++;

}

}

delay\_ms(SECOND);

}

else if (ptr == UNLOCKED)

{

if(flag == 0)

{

print\_screen(ulock1, ulock2);

flag++;

}

char in = keypad\_Getkey();

if (in != 0xFF)

{

flag = 0;

if(nums[in]!='#') // reset pin combination

{

next = 1;

}

else

{

next = 2;

}

}

delay\_ms(SECOND);

}

else if (ptr == RESET)

{

if(flag==0)

{

print\_screen(res1, res2);

flag++;

}

if(i == PASSLEN)

{

i = 0;

flag = 0;

next = 1;

}

else

{

char in = keypad\_Getkey();

if (in != 0xFF)

{

if (!(nums[in] == '\*' || nums[in] == '#')) //accept only valid numbers

{

write\_char\_LCD(nums[in]);

key[i] = nums[in];

i++;

}

}

delay\_ms(SECOND);

}

}

else if (ptr == TIMEOUT)

{

print\_screen(time1, time2);

delay\_ms(10 \* SECOND);

next = 1;

flag = 0;

tries = 0;

}

ptr = ptr->Next[next];

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*

\* delay.h

\* this holds the software delay functions as well as the funcion to change the clock

\* frequency

\* Created on: Apr 19, 2020

\* Author: nicks

\*/

#ifndef DELAY\_H\_

#define DELAY\_H\_

#define FREQ\_1\_5MHz CS\_CTL0\_DCORSEL\_0

#define FREQ\_3MHz CS\_CTL0\_DCORSEL\_1

#define FREQ\_6MHz CS\_CTL0\_DCORSEL\_2

#define FREQ\_12MHz CS\_CTL0\_DCORSEL\_3

#define FREQ\_24MHz CS\_CTL0\_DCORSEL\_4

#define FREQ\_48MHz CS\_CTL0\_DCORSEL\_5

void delay\_ms(uint32\_t delay);

void delay\_us(int us, int freq);

void set\_DCO(int val);

#endif /\* DELAY\_H\_ \*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*

\* delay.c

\*

\* Created on: Apr 19, 2020

\* Author: nicks

\*/

#include "msp.h"

#include "delay.h"

#define DELAY100MS 300000u // DELAY \* 1 mS assuming 3 MHz clock

#define PORT\_MCLK\_OUT (P4)

#define BITM\_MCLK\_OUT (BIT3)

#define DELAY100MS 300000u // DELAY \* 1 mS assuming 3 MHz clock

#define PORT\_MCLK\_OUT (P4)

#define BITM\_MCLK\_OUT (BIT3)

/\* @brief : sets the MCLK using the digitally controlled oscillator

\* to 24MHz and the SMCLK to 12MHz

\* @param : frequency bitmask (macro defined in dco.h)

\* @return: N/A

\*/

void delay\_ms(uint32\_t delay)

{

uint32\_t i;

for(i=0; i<(delay<<6); i++);

}

void set\_DCO(int val)

{

/\* Setting up Master CLK based on freq\_sel \*/

CS->KEY = CS\_KEY\_VAL; // unlocks CS

CS->CTL0 = val;

CS->CTL1 = *0b011*;

CS->KEY = 0; // locks CS

}

void delay\_us(int us, int freq)

{

int count;

/\* tuned for 40ms \*/

if (freq == FREQ\_1\_5MHz)

{

count = us / 7;

}

/\* tuned for 40us \*/

else if (freq == FREQ\_3MHz)

{

count = us / 6;

}

/\* tuned for 40ms \*/

else if (freq == FREQ\_6MHz)

{

count = (us \* 10) / 18;

}

/\* tuned for 40us \*/

else if (freq == FREQ\_12MHz)

{

count = us;

}

/\* tuned for 40ms \*/

else if (freq == FREQ\_24MHz)

{

count = (us \* 100) / 46;

}

else if (freq == FREQ\_48MHz)

{

count = (us \* 48) / 10;

}

while( count > 0)

{

\_\_delay\_cycles(1);

count-=1;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*

\* keypad.h

\*

\* Created on: Apr 26, 2020

\* Author: nicks

\*/

#ifndef KEYPAD\_H\_

#define KEYPAD\_H\_

#include <stdint.h>

#define COL\_0 BIT0

#define COL\_1 BIT1

#define COL\_2 BIT2

#define ROW\_0 BIT3

#define ROW\_1 BIT4

#define ROW\_2 BIT5

#define ROW\_3 BIT6

void keypad\_Init(void);

uint8\_t keypad\_Getkey(void);

#endif /\* KEYPAD\_H\_ \*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "keypad.h"

#include "msp.h"

#include "delay.h"

#include "lcd.h"

void keypad\_Init(void) {

P2->SEL0 &= ~(ROW\_0 | ROW\_1 | ROW\_2 | ROW\_3); // Sets P5.0-P5.2 to GPIO (COL)

P2->SEL1 &= ~(ROW\_0 | ROW\_1 | ROW\_2 | ROW\_3);

P2->DIR &= ~(ROW\_0 | ROW\_1 | ROW\_2 | ROW\_3); // Sets inputs with a pulldown resistor

P2->REN |= (ROW\_0 | ROW\_1 | ROW\_2 | ROW\_3);

P2->OUT &= ~(ROW\_0 | ROW\_1 | ROW\_2 | ROW\_3);

P5->SEL0 &= ~(COL\_0 | COL\_1 | COL\_2); // Sets P2.3-P2.6 to GPIO (ROW)

P5->SEL1 &= ~(COL\_0 | COL\_1 | COL\_2);

P5->DIR |= (COL\_0 | COL\_1 | COL\_2); // Sets P2.3-P2.6 to outputs

}

/\*

\* If a key is pressed, it returns that key value 0-9. \* is 10, # is 12

\* If no key is pressed, it returns 0xFF

\*/

uint8\_t keypad\_Getkey(void) {

uint8\_t row, col, key;

/\* check to see any key pressed \*/

P5->OUT |= (COL\_0 | COL\_1 | COL\_2); // drive all column pins high

\_delay\_cycles(25); // wait for signals to settle

row = P2->IN & (ROW\_0 | ROW\_1 | ROW\_2 | ROW\_3); // 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 cols

P5->OUT &= ~(COL\_0 | COL\_1 | COL\_2);

// shift a 1 into the correct column depending on which to turn on

P5->OUT |= (COL\_0 << col);

\_delay\_cycles(25); // wait for signals to settle

row = P2->IN & (ROW\_0 | ROW\_1 | ROW\_2 | ROW\_3); // mask only the row pins

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

}

P5->OUT &= ~(COL\_0 | COL\_1 | COL\_2); // drive all columns low

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

// rows are read in binary, so powers of 2 (8,16,32,64)

if (row == 8) row = 1;

if (row == 16) row = 2;

if (row == 32) row = 3;

if (row == 64) 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;

return key;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*

\* lcd.h

\*

\* Created on: Apr 19, 2020

\* Author: nicks

\*/

#ifndef LCD\_H\_

#define LCD\_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 clear\_LCD();

void home\_LCD();

void LCD\_command(unsigned char command);

void LCD\_init(void);

void write\_char\_LCD(char data);

void write\_string\_LCD(char str[]);

#endif /\* LCD\_H\_ \*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*

\* lcd.c

\*

\* Created on: Apr 19, 2020

\* Author: nicks

\*/

#include "msp.h"

#include "lcd.h"

#include "delay.h"

void LCD\_init(void)

{

P4->DIR = 0xFF; /\* make P4 pins output for data and controls \*/

delay\_us(30000,FREQ\_3MHz); /\* initialization sequence \*/

LCD\_nibble\_write(0x30, 0);

delay\_us(10000,FREQ\_3MHz);

LCD\_nibble\_write(0x30, 0);

delay\_us(1000,FREQ\_3MHz);

LCD\_nibble\_write(0x30, 0);

delay\_us(1000,FREQ\_3MHz);

LCD\_nibble\_write(0x20, 0); /\* use 4-bit data mode \*/

delay\_us(1000,FREQ\_3MHz);

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 \*/

delay\_us(1000,FREQ\_3MHz);

P4->OUT = data; /\* clear E \*/

P4->OUT = 0;

}

void LCD\_command(unsigned char command)

{

LCD\_nibble\_write(command & 0xF0, 0); /\* upper nibble first \*/

LCD\_nibble\_write(command << 4, 0); /\* then lower nibble \*/

if (command < 4)

delay\_us(4000,FREQ\_3MHz); /\* commands 1 and 2 need up to 1.64ms \*/

else

delay\_us(1000,FREQ\_3MHz); /\* all others 40 us \*/

}

void clear\_LCD()

{

LCD\_command(0x01);

}

void home\_LCD()

{

LCD\_command(0x80);

}

void write\_char\_LCD(char data)

{

LCD\_nibble\_write(data & 0xF0, RS); /\* upper nibble first \*/

LCD\_nibble\_write(data << 4, RS); /\* then lower nibble \*/

delay\_us(1000,FREQ\_3MHz);

}

void write\_string\_LCD(char str[])

{

int i=0;

while(str[i])

{

write\_char\_LCD(str[i]);

i++;

}

}

**References**

[MSP432 datasheet](http://www.ti.com/lit/ds/symlink/msp432p401r.pdf?ts=1588064064864)