

Lab 03

# Keypad, Seven Segment, and LCD

Logan Barber, Ian Nail

February 11, 2022

*Ian Nail*

*Logan Barber*

ME-4370 - Stephen Canfield

# 1 Executive Summary

In this lab, the objective is to write a program to use hardware pushbuttons for a passcode program that requires the user to make a passcode and then enter that passcode to unlock the recipe.

The minimum objectives were:

1. Prompt the user to enter a 4 digit passcode
2. The passcode is entered using hard wired pushbuttons.
3. Save the 4 numbers into memory
4. Once the passcode has been entered the safe needs to be closed. For a recipe book, this can be a display, "Recipe locked".
5. Prompt the user to press any switch to unlock the safe.
6. Prompt the user enter their passcode
7. Check the passcode, open if correct, do not open if incorrect.
8. Allow the user to enter the entire passcode before checking it.
9. Allow the user up to three attempts, if they fail on the third attempt, display failure message, and end the program without opening.
10. When the passcode succeeds, you can continue to the recipe.

The seven segment 4 display for the timer and the LCD display program from Lab02 was carried over as the basis for implementing the keypad into a better rounded project.

To get an input from the keypad, the button matrix is scanned one row and column at a time as the program searches for a button press. Two 'for loops' are used to iteratively do the row and column search. The keypad has 16 buttons, so 4 rows and 4 columns needed 8 bits. PORTC was used with 4 bits as inputs and 4 bits as outputs. The inputs bits had the pull-up resistor enabled. The pull-up resistors were used for reading active lows when a button press occurs. The 16 buttons are scanned so quickly that it is very challenging to not have a button press registered. Debouncing code is used to hold the processor in an infinite loop until the button is released. This prevents a single button press from being interpreted as multiple button presses. A 'do while' loop is used for setting the password and another 'do while' loop used for checking the entered password matched the password that had been set. After three failed password attempts the program gives a message and fails to continue. While the password is being set or entered, the LCD screen shows the characters from the button presses so the user can see that each button was registered. The pound sign is used to end the password that is entered. If a password is successful the program will continue on to give the recipe instructions and the timer becomes available to use.

AutoCad was used to draw up the circuit diagram of all the connections, components, and devices used.



## 2 Source Code

```
0  /* AUTHORS: A. LOGAN BARBER; IAN NAIL
   * FILE NAME: Lab03.ino
   * LAST UPDATED: 28 JANUARY 2022
   *
   * PURPOSE: THIS FILE IS THE MAIN FILE FOR DISPLAYING A RECIPE ON AN LCD AND
   *          BEING ABLE TO SCROLL USING TWO BUTTONS.
   *          THIS FILE ALSO UTILIZES AND 4-7 SEGMENT DISPLAY AS A 10 MINUTE TIMER
   *          AND A KEYPAD FOR PASSWORD PROTECTION
   */
6
8  /*
   * BUTTON0:
   * DIGITAL PIN: 22
   * PORT: A
   * PORT PIN: 0
   */
14
16 /*
   * BUTTON1:
   * DIGITAL PIN: 23
   * PORT: A
   * PORT PIN: 1
   */
20
22 /*
   * BUTTON2:
   * DIGITAL PIN: 24
   * PORT: A
   * PORT PIN: 2
   */
28
30 /*
   * 7 - SEGMENT DECODER
   *
   * D0:
   *     DIGITAL PIN: 53
   *     PORT: B
   *     PORT PIN: 0
   *
   * D1:
   *     DIGITAL PIN: 52
   *     PORT: B
   *     PORT PIN: 1
   *
   * D2:
   *     DIGITAL PIN: 51
   *     PORT: B
   *     PORT PIN: 2
   *
   * D3:
   *     DIGITAL PIN: 50
   *     PORT: B
   *     PORT PIN: 3
   */
52
54 /*
   * DEMULTIPLEXER
```

```

56  *
57  * A:
58  *     DIGITAL PIN: 10
59  *     PORT: B
60  *     PORT PIN: 4
61  *
62  * B:
63  *     DIGITAL PIN: 11
64  *     PORT: B
65  *     PORT PIN: 5
66  */
67
68  /*
69  * KEY PAD:
70  *
71  * PIN 1:
72  *     DIGITAL PIN: 37
73  *     PORT: C
74  *     PORT PIN: 0
75  *
76  * PIN 2:
77  *     DIGITAL PIN: 36
78  *     PORT: C
79  *     PORT PIN: 1
80  *
81  * PIN 3:
82  *     DIGITAL PIN: 35
83  *     PORT: C
84  *     PORT PIN: 2
85  *
86  * PIN 4:
87  *     DIGITAL PIN: 34
88  *     PORT: C
89  *     PORT PIN: 3
90  *
91  * PIN 5:
92  *     DIGITAL PIN: 33
93  *     PORT: C
94  *     PORT PIN: 4
95  *
96  * PIN 6:
97  *     DIGITAL PIN: 32
98  *     PORT: C
99  *     PORT PIN: 5
100  *
101  * PIN 7:
102  *     DIGITAL PIN: 31
103  *     PORT: C
104  *     PORT PIN: 6
105  *
106  * PIN 8:
107  *     DIGITAL PIN: 30
108  *     PORT: C
109  *     PORT PIN: 7
110  */
111
112  // INCLUDE LIBRARIES
113  #include <LiquidCrystal_I2C.h>
114  #include <stdio.h>

```

```

116 #define TRUE 0x01
117 #define FALSE 0x00
118 // DEFINE MACROS FOR LCD SERIAL
119 #define ADDRESS 0x27
120 #define LCDCOLS 16
121 #define LCDROWS 2
122 // TIMER PARAMTERS
123 uint8_t buttonState = 0;
124 uint8_t minutes = 6; //start time -> CAN CHANGE TO WHATEVER TIME YOU WANT
125 uint8_t seconds = 30; //start time -> CAN CHANGE TO WHATEVER TIME YOU WANT
126 uint8_t totalMinutes = 0;
127 uint8_t minutesTens = 0;
128 uint8_t minutesOnes = 0;
129 uint8_t secondsTens = 0;
130 uint8_t secondsOnes = 0;
131 uint8_t secondsTemp = 0;
132 float totalSeconds = minutes*60 + seconds;
133 float totalMilliseconds = totalSeconds*1000;
134 float totalMicroseconds = totalMilliseconds*1000;
135 // DEFINE NUMBERS FOR 7 SEGMENT DISPLAY
136 #define ZERO 0x00
137 #define ONE 0x01
138 #define TWO 0x02
139 #define THREE 0x03
140 #define FOUR 0x04
141 #define FIVE 0x05
142 #define SIX 0x06
143 #define SEVEN 0x07
144 #define EIGHT 0x08
145 #define NINE 0x09
146 // DEFINE KEYPAD ITEMS
147 #define ONEPAD 0x77 // 0b01110111
148 #define TWOPAD 0x7B // 0b01111011
149 #define THREEPAD 0x7D // 0b01111101
150 #define APAD 0x7E // 0b01111110
151 #define FOURPAD 0xB7 // 0b10110111
152 #define FIVEPAD 0xBB // 0b10111011
153 #define SIXPAD 0xBD // 0b10111101
154 #define BPAD 0xBE // 0b10111110
155 #define SEVENPAD 0xD7 // 0b11010111
156 #define EIGHTPAD 0xDB // 0b11011011
157 #define NINEPAD 0xDD // 0b11011101
158 #define CPAD 0xDE // 0b11011110
159 #define STARPAD 0xE7 // 0b11100111
160 #define ZEROPAD 0xEB // 0b11101011
161 #define POUNDPAD 0xED // 0b11101101
162 #define DPAD 0xEE // 0b11101110
163 // Keypad Parameters
164 const byte ROWS = 4; // number of rows on keypad
165 const byte COLS = 4; // number of columns on keypad
166 char keys[ROWS][COLS] = { // The buttons on the keypad
167 { '1', '2', '3', 'A' },
168 { '4', '5', '6', 'B' },
169 { '7', '8', '9', 'C' },
170 { '*', '0', '#', 'D' } };

```

```

176 int passwordSucceed = FALSE;
char keyResult = 'F';
int password = '0';

178 // DEFINE LETTERS FOR 7 SEGMENT DISPLAY
180 #define ZERO 0x00
#define ONE 0x01
182 #define TWO 0x02
#define THREE 0x03
184 #define FOUR 0x04
#define FIVE 0x05
186 #define SIX 0x06
#define SEVEN 0x07
188 #define EIGHT 0x08
#define NINE 0x09

190 // DEFINE THE DISPLAY SELECTION NUMBERS
192 const uint8_t D1 = 0x0F; // 0b00001111
const uint8_t D2 = 0x1F; // 0b00011111
194 const uint8_t D3 = 0x2F; // 0b00101111
const uint8_t D4 = 0x3F; // 0b00111111
196 const uint8_t arrD[4] = {D1, D2, D3, D4};

198 // MESSAGE TO PRINT
char password0[] = "Enter Password: ";
200 char message0[] = "Double Chocolate";
char message1[] = "Flower Brownies";
202 char message2[] = "1/2 Cup";
char message3[] = "Unsalted Butter";
204 char message4[] = "1 Gram";
char message5[] = "Flower";
206 char message6[] = "1/4 Cup ";
char message7[] = "Chocolate Chips";
208 char message8[] = "1 Tablespoon";
char message9[] = "Molasses";
210 char message10[] = "1 Teaspoon";
char message11[] = "Vanilla Extract";
212 char message12[] = "2 Large Eggs";
char message13[] = " ";
214 char message14[] = "1/4 Teaspoon";
char message15[] = "Kosher Salt";
216 char message16[] = "3/4 Cup All-";
char message17[] = "Purpose Flour";
218 char message18[] = "Bake for 10";
char message19[] = "minutes";
220 const uint8_t msgArrSize = 20;
char* msgArr[msgArrSize] = {message0, message1, message2, message3, message4,
222     message5, message6, message7, message8, message9, message10, message11,
    message12,
    message13, message14, message15, message16, message17, message18, message19
};

224 // INDEX VARIABLES
226 uint8_t index = 0; // HOLDS INDEX FOR MESSAGE
uint8_t i = 0; // HOLDS INDEX IN for LOOPS FOR SCROLLING
228 uint8_t t = 0; // HOLDS INDEX IN for LOOP FOR THE TIMER

230 // CREATE LiquidCrystal OBJECT
LiquidCrystal_I2C lcd(ADDRESS, LCDCOLS, LCDROWS);

232

```

```

234 // RUN THIS PROGRAM
void setup()
{
236     // INITIALIZE THE LCD SCREEN
    lcd.begin();
238     // SETUP BUTTON PINS AS INPUTS
    DDRA = 0x00; // 0b00000000

240     // ENABLE INTERNAL PULL-UP RESISTOR FOR BUTTONS
    PORTA = 0x07; // 0b00000111

244     // SETUP PORT B AS OUTPUT FOR THE LCD
    DDRB = 0x7F; // 0b01111111
    PORTB = 0x00;

248     // SET UP PORT C AS OUTPUT AND INPUT
    DDRC = 0x0F;
250     PORTC = 0xFF; // ENABLE PULL UP RESISTOR FOR INPUTS ON PORTC

252     // CALCULATE INDIVIDUAL DIGITS
    totalMinutes = totalSeconds/60;
254     minutesTens = totalMinutes/10;
    minutesOnes = totalMinutes%10;
256     secondsTemp = int(totalSeconds)%60;
    secondsTens = secondsTemp/10;
258     secondsOnes = secondsTemp%10;

260     // This function sets the password and lockes the recipe till the password
    is entered
    passwordSetandLock();
262 }
// LOOP FOREVER
264 void loop()
{
266     // IF BUTTON0 IS LOW SCROLL DOWN
    if((PINA & 0x07) == 0x06)
268     {
        // DEBOUNCE BUTTON0
270        delay(100);
        if((PINA & 0x07) == 0x06)
272        {
            scroll_down();
274        }
    }

276    // ELSE IF BUTTON1 IS LOW SCROLL DOWN
    if((PINA & 0x07) == 0x05)
278    {
        // DEBOUNCE BUTTON2
280        delay(100);
        if((PINA & 0x07) == 0x05)
282        {
            scroll_up();
284        }
    }

286    // IF BUTTON 2 IS LOW CHANGE THE BUTTON STATE
    else if((PINA & 0x07) == 0x03)
288    {
290        delay(100);

```



```

292         if((PINA & 0x07) == 0x03)
293         {
294             switch(buttonState)
295             {
296                 case 0:
297                     buttonState = 1;
298                     break;
299
300                 case 1:
301                     buttonState = 2;
302                     break;
303
304                 case 2:
305                     buttonState = 0;
306
307                     // RESET TIME
308                     totalSeconds = minutes*60 + seconds;
309                     totalMilliseconds = totalSeconds*1000;
310                     totalMicroseconds = totalMilliseconds*1000;
311
312                     // CALCULATE INDIVIDUAL DIGITS
313                     totalMinutes = totalSeconds/60;
314                     minutesTens = totalMinutes/10;
315                     minutesOnes = totalMinutes%10;
316                     secondsTemp = int(totalSeconds)%60;
317                     secondsTens = secondsTemp/10;
318                     secondsOnes = secondsTemp%10;
319                     break;
320             }
321         }
322     }
323
324     // RUN TIMER IF BUTTON STATE IS IN STATE 1
325     if(buttonState == 1)
326     {
327         // TIME CALCULATIONS
328         totalMicroseconds = totalMicroseconds - 2000; //totalMilliseconds++'
329     for stopwatch
330         totalMilliseconds = totalMicroseconds/1000;
331         totalSeconds = (totalMilliseconds/1000+1);
332
333         // CALCULATE INDIVIDUAL DIGITS
334         totalMinutes = totalSeconds/60;
335         minutesTens = totalMinutes/10;
336         minutesOnes = totalMinutes%10;
337         secondsTemp = int(totalSeconds)%60;
338         secondsTens = secondsTemp/10;
339         secondsOnes = secondsTemp%10;
340     }
341
342     // TIMER
343     for(t = 0; t < 4; ++t)
344     {
345         switch(t)
346         {
347             case 0:
348                 PORTB &= 0x0F;
349                 PORTB |= arrD[t];
350                 seven_seg_writeNumber(minutesTens);
351                 delayMicroseconds(500);

```

```

352         break;

353     case 1:
354         PORTB &= 0x0F;
355         PORTB |= arrD[t];
356         seven_seg_writeNumber(minutesOnes);
357         PORTB |= 0x40;
358         delayMicroseconds(500);
359         break;

360     case 2:
361         PORTB &= 0x0F;
362         PORTB |= arrD[t];
363         seven_seg_writeNumber(secondsTens);
364         delayMicroseconds(500);
365         break;

366     case 3:
367         PORTB &= 0x0F;
368         PORTB |= arrD[t];
369         seven_seg_writeNumber(secondsOnes);
370         delayMicroseconds(500);
371         break;

372     default:
373         break;
374 }
375 }
376 }
377 }
378 }
379 }
380
381 /*
382  * TYPE: FUNCTION
383  * NAME: scroll_up
384  * RETURN: void
385  * NUMBER OF PARAMETERS: 2
386  * PARAMETER NAMES: char* messagePtr, uint8_t sizeOfArray
387  * PURPOSE: THIS FUNCTION SCROLLS THROUGH THE RECEIPE DISPLAYED ON THE LCD
388  */
389 void scroll_up()
390 {
391     // DECREMENT INDEX BY ONE
392     index -= 2;
393
394     // CHECK THE BOUNDS OF INDEX (REMEMBER index IS UNSIGNED)
395     if(index > (msgArrSize - 2))
396         index = 0;
397
398     // CLEAR THE LCD SCREEN AND PRINT MESSAGES TO THE LCD
399     lcd.clear();
400     for(i = 0; i < 2; ++i)
401     {
402         lcd.setCursor(0, i);
403         delayMicroseconds(500);
404         lcd.print(msgArr[index + i]);
405         delayMicroseconds(500);
406     }
407 }
408
409 /*
410  * TYPE: FUNCTION

```

```

412  * NAME: scroll_down
413  * RETURN: void
414  * NUMBER OF PARAMETERS: 2
415  * PARAMETER NAMES: char* messagePtr, uint8_t sizeOfArray
416  * PURPOSE: THIS FUNCTION SCROLLS THROUGH THE RECEIPE DISPLAYED ON THE LCD
417  */
418  void scroll_down()
419  {
420      // INCREMENT INDEX
421      index += 2;
422
423      // CHECK THE BOUNDS OF INDEX
424      if(index > (msgArrSize - 2))
425          index = msgArrSize - 2;
426
427      // CLEAR THE LCD SCREEN AND PRINT MESSAGES TO THE LCD
428      lcd.clear();
429      for(i = 0; i < 2; ++i)
430      {
431          lcd.setCursor(0, i);
432          delayMicroseconds(1000);
433          lcd.print(msgArr[index + i]);
434          delayMicroseconds(1000);
435      }
436
437  /*
438  * TYPE: FUNCTION
439  * NAME: seven_seg_writeNumber
440  * RETURN: void
441  * NUMBER OF PARAMETERS: 1
442  * PARAMETER NAMES: int x
443  * PURPOSE: THIS FUNCTION PICKS THE NUMBER FOR THE LCD
444  */
445  void seven_seg_writeNumber(int x) //changes value of number
446  {
447      switch(x)
448      {
449          case 1:
450              PORTB &= 0b00110000;
451              PORTB |= ONE;
452              break;
453
454          case 2:
455              PORTB &= 0b00110000;
456              PORTB |= TWO;
457              break;
458
459          case 3:
460              PORTB &= 0b00110000;
461              PORTB |= THREE;
462              break;
463
464          case 4:
465              PORTB &= 0b00110000;
466              PORTB |= FOUR;
467              break;
468
469          case 5:
470              PORTB &= 0b00110000;

```

```

472         PORTB |= FIVE;
         break;

474     case 6:
         PORTB &= 0b00110000;
476         PORTB |= SIX;
         break;

478     case 7:
         PORTB &= 0b00110000;
480         PORTB |= SEVEN;
482         break;

484     case 8:
         PORTB &= 0b00110000;
486         PORTB |= EIGHT;
         break;

488     case 9:
         PORTB &= 0b00110000;
490         PORTB |= NINE;
492         break;

494     default:
         PORTB &= 0b00110000;
496         PORTB |= ZERO;
         break;

498 }
}

500
501 /*
502  * TYPE: FUNCTION
503  * NAME: keyReturn
504  * RETURN: char
505  * NUMBER OF PARAMETERS: 1
506  * PARAMETER NAMES: void
507  * PURPOSE: THIS returns the character value of the button pressed on the keypad
508  */
509 char keyReturn(void)
510 {
511     /* for loop execution */
512     byte outputROWS[ROWS] = {0b00000001,0b00000010,0b00000100,0b00001000}; //
    last 4 bits of PORTC that will be toggled to check if a button is pressed
    byte inputCOLS[COLS] = {0b00010000,0b00100000,0b01000000,0b10000000}; //
    first 4 bits of PORTC that will be read from PINC to see if the pullup
    resistor has been grounded
514     byte a = 0;
    byte b = 0;
516     int valuePINC_B = 0;

518     for (a = 0; a < 4; a = a+1) //first 'for' loop to check the rows of the
    button matrix
    {
520         PORTC ^= outputROWS[a]; // toggle the bit low

522         for (b = 0; b < 4; b = b+1) //second for loop to check the columns of
    the button matrix
        {
524             valuePINC_B = (PINC & inputCOLS[b]) >> (b+4); // get value to check
    if the bit is low (low means a button has been pressed at row a and column b

```

```

526         if (valuePINC_B != 1) // if the value is not high that means a
button press is detected
        {
528             keyResult = keys[a][b];
            delay(15); //Debouncing code
530             while(1 != (PINC & inputCOLS[b]) >> (b+4));
            {
532             }
        }
534     }

536     PORTC ^= outputROWS[a]; //toggle the bit high
    }

538     return keyResult;
540 }

542 /*
* TYPE: FUNCTION
544 * NAME: passwordSetandLock
* RETURN: void
546 * NUMBER OF PARAMETERS: 0
* PARAMETER NAMES: n/a
548 * PURPOSE: THIS SETS AND LOCKES THE RECIPE TILL THE CORRECT PASSWORD IS ENTERED
. THREE MAX TRIES TO ENTER CORRECT PASSWORD
*/
550 void passwordSetandLock()
{
552     // PROMPT USER FOR PASSWORD
    lcd.print(password0);
554     lcd.setCursor(0, 1);

556     // GET USER PASSWORD
    do
558     {
        keyReturn();
560        passwordSucceed = 0;

562        if (keyResult != 'F')
        {
564            password = password + keyResult;
            lcd.print(keyResult);
566            if (keyResult == '#')
            {
568                passwordSucceed = TRUE;
            }

570            keyResult = 'F';
572        }
    } while(passwordSucceed == FALSE);

574     lcd.clear();
576     lcd.print("RECIPE LOCKED");
    lcd.setCursor(0, 1);

578     //GET USER TO ENTER PASSWORD TO UNLOCK RECIPE
580     int passwordTest = '0';
    int passwordTries = 0;
582     do

```

```

584     {
        keyReturn();
        passwordSucceed = 0;

586

588         if (keyResult != 'F')
        {
590             passwordTest = passwordTest + keyResult;
            lcd.print(keyResult);
592             if ((keyResult == '#') && (passwordTest == password))
            {
594                 passwordSucceed = TRUE;
            }
596             if ((keyResult == '#') && (passwordTest != password))
            {
598                 passwordSucceed = FALSE;
                passwordTest = '0';
600                 lcd.clear();
                lcd.print("Try Again");
602                 lcd.setCursor(0, 1);
                passwordTries++;
604                 if(passwordTries >= 3)
                {
606                     lcd.clear();
                        lcd.print("NO MORE TRIES");
608                     lcd.setCursor(0, 1);
                        lcd.print("LEFT! ");
610                     while(1)
                        {
612                         }
                    }
                }
614                 keyResult = 'F';
            }
616         } while(passwordSucceed == FALSE);

618         // PRINT MESSAGE
620         lcd.clear();
        lcd.print(msgArr[0]);
622         lcd.setCursor(0, 1);
        lcd.print(msgArr[1]);
624     }

```

Lab03.ino