

## Keypad System

---

*Matthew R., Sean J., and Brendan R.*  
Rowan University

December 19, 2018

### 1 Abstract

Within the Final project, students created a keypad security system using momentary push button switches, 7 segment architecture, and a speaker to provide the control and response portion of a security system.

### 2 Introduction

Micro controllers play an important part within the world of controlling systems, and within the security world this is paramount. One of the simplest things to do in terms of securing something is preventing access to something. While the mechanism that does the unlocking and locking can change from system to system, they all work towards the goal of only allowing access to certain people. A good example of this within the modern day world are keypads. Keypads can be used anywhere, from locking entrances on doors to keeping safes secure, keypad locks are a simple concept meant to maintain security.

### 3 Background

#### 3.1 Momentary Push Button Switch

The keypad used in this project contains an old style of tactile switches that when pressed, click to give a physical feedback of a button press, while also signaling audibly when electrical contact is made. For momentary switches they send a signal when pressed and then when depressed, the connection is severed, giving a momentary signal. These switches differ from other types, such as toggle switches. Toggle switches when pressed will stay pressed until it is pressed again to release.

### 3.2 7seg values

An important part of using a keypad security system is being able to set values and compare them to check for a correct or incorrect input. Within this project, 7seg values were set to each of the buttons on the keypad. With each button set to a 7seg value, the code lock can be stored within the micro controller as a set of values representing each button press. Now the micro controller can simply compare any series of input values to the lock's code. A matching input will unlock the device, while a non-matching input will maintain the current locked state. 74HC595 shift registers were used to reduce the number of wires between the 7-seg display and incorporate serial communication. The benefit of using the numbers stored as 7seg allows easy display of the input numbers on a digital number screen using that format. This allows the user to keep track of the inputted numbers.

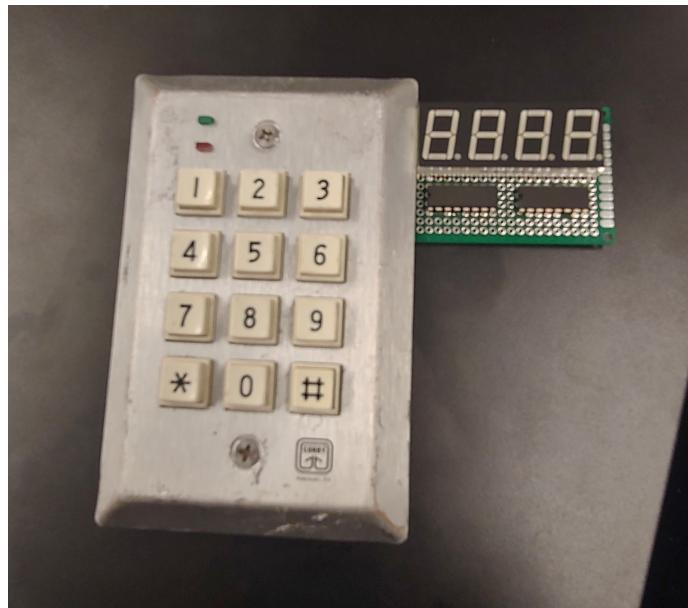


Figure 1: Keypad Components

## 4 Evaluation and Results

### 4.1 Evaluation

The way the keypad works currently is that to set the pass code for the keypad, the user must input a pass code embedded within the code. This is merely a placeholder pass code that will be overwritten once the embedded code is inputted and overwritten. The universal pass code will be provided to the user so that they may set their

own pass code. Putting in a correct pass code, a correct pass code being an input that matches the stored pass code within the micro controller, will have the key pad give an output that is relevant to the system it is attached to, for example in a keypad lock it engage the motor to move a deadbolt.

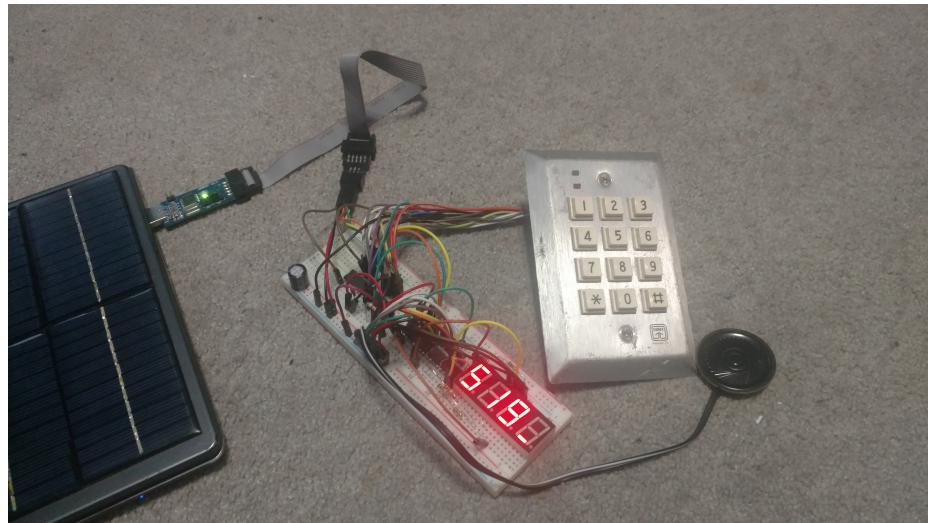


Figure 2: Breadboard Demo

The keypad system is equipped with a speaker to give audible feedback to the user, such as when a button has been pressed. The speaker also serves to let the user know when they've inputted a correct or incorrect pass code. If the inputted code is incorrect 4 consecutive times, the speaker will emit a constant beep. The audible aspect of the keypad will improve the user experience as the sight of the number appearing on the digital display as well as the sound provides confirming sensory data to the user.

## 4.2 Results

Within the breadboard demo, the device was then tested with an input that was not the same as the set password. The micro controller recognized it as incorrect, and outputted the designated noise for it (two low-pitched beeps). Afterwards, the device was tested with the same input as the correct password. Upon receiving this input, the device recognized it as the set password, and outputted the designated noise for it (one low pitched beep, one high-pitched beep). The next test was to input four consecutive incorrect codes into it. Upon putting in the four incorrect codes, the speaker within the system began to output a constant alarm. This tone ceased once the correct password was inserted.

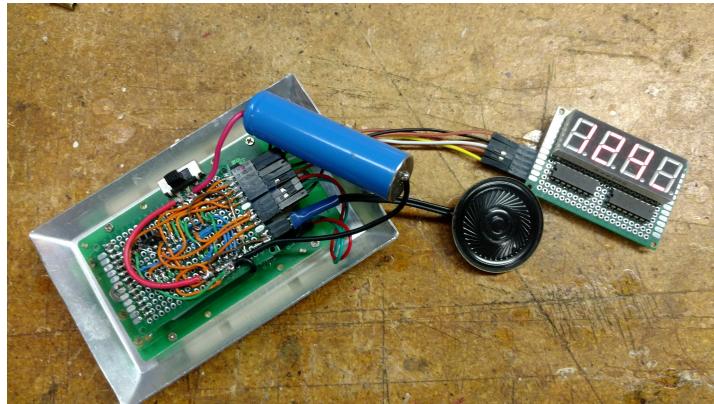


Figure 3: Keypad System with Lithium Battery

## 5 Discussion/Conclusions

For design choices within the lab, feedback was an important part for using the keypad, the feedback from the board can be seen with both audio and visual signals, through the 7 segment display and the speaker, this way we have an affirmative noise and the 7 segment display says a message. These are two ways the machine communicates with a person. The module senses by taking in multiple inputs from the keypad, namely 4 digits, and sends that information to the brain. Then the module computes the input and decides whether or not the unit should open, or in our case the speaker yells at you.

## 6 Cost analysis

ITEM	QTY	PRICE/EA
ATTINY88-pu	1	\$0.73
1/4 Carbon Film Res 6800Ω	8	\$0.00
4x7 segment display	1	\$0.61
470 uF capacitor	1	\$0.00
Protoboard	1	\$0.31
74HC595	2	\$0.06
Keypad	1	\$1.00
TP4056	1	\$0.60
TOTAL		\$3.39

## 7 Appendix

### 7.1 Appendix A: Architecture

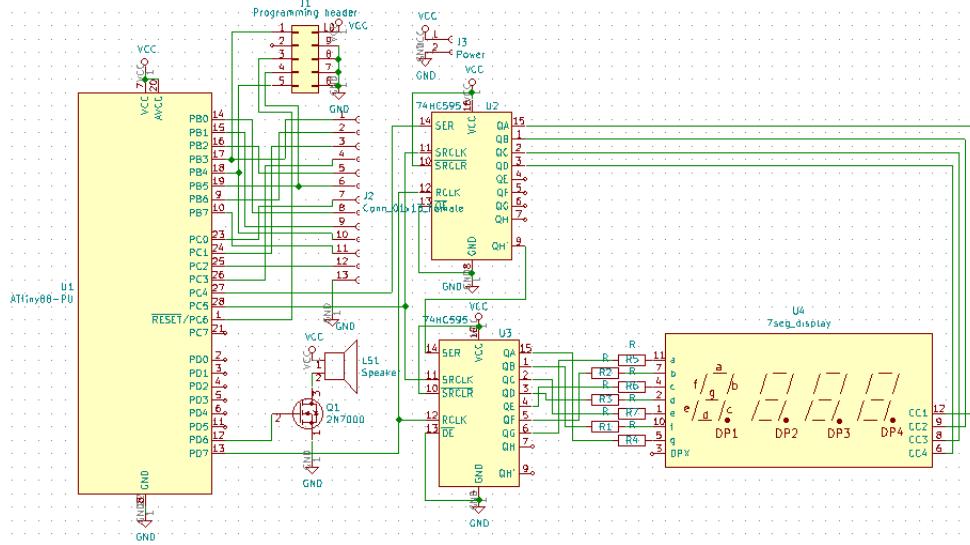


Figure 4: System Schematic Architecture

### 7.2 Appendix B: Spiral Design

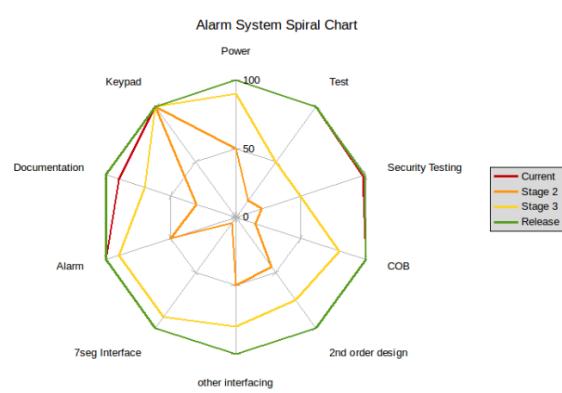


Figure 5: Spiral Design Progress

### 7.3 Appendix C: Code

To access the code itself, find it on Github at the following link: <https://github.com/RU09342-F18/intro-to-embedded-final-project-black-mesa/blob/master/ATtiny88-pu/main.c>

```

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
/*
 * Pin definitions needed, 02 available
 */
8//RS RS //open
7//R0 R0 //open
7//R1 TX //open
8//RXD2 Door lock
9//TXD4 LED6
13//VCC VCC
12//GND GND
13//PB6 6 PCINT6
14//PB5 5 PCINT5
15//PB5 LED5
16//PB6 Siren
17//PB7 1 PCINT7
18//PB8 0 PCINT8
19//PB9 2 PCINT9
20//PB10 2 RX 2
21//PC4 DAT 3
22//PC2 * PCINT10
23//PC2 9 PCINT9
24//PC1 9 PCINT8
25//PC1 8 PCINT7
26//GND GND
27//ref VCC
28//ref VCC
29//PB5 5 PCINT5
30//PB4 4 PCINT4
31//PB3 4 PCINT3
32//PB2 2 PCINT2
33//PB1 1 PCINT1
34//void Timer_Init(void){
35    TMR0R = (1<<OC1A0);
36    TMR0R |= (1<<OC0A0);
37    TCCR0B = (1<<CS00);
38
39    //TMR0A1 = (1<<OC11); //timer 1 OVF interrupt enable
40}*//*
41uint8_t receivechar[] = {0x00,0x10,0x10,0x10};
42uint8_t keypadctr = 0,regctr = 0x10, attempts=0x66,icounter = 0;
43uint8_t code1 = {0x00,0x00,0x00,0x00};
44
45/*SRCDIMER1_COMP_A_vect{
46    TCCR1B = -1<<CS11; //timer 1 clockdiv disable
47}*//*
48void sand(uint8_t input){
49    for(uint8_t i = 0;i<4;i++){
50        if(input & (1<<i)){
51            PORTC |= (1<<PC4);
52        }else
53            PORTC |= -(1<<PC4);
54        PORTC |= -(1<<PC3);
55        input = Input>>1;
56        PORTC |= -(1<<PC2);
57        PORTC |= -(1<<PC1);
58    }
59}
60void latch(void){
61    PORTD |= -(1<<PD7);
62    PORTD &= -(1<<PD7);
63}
64uint8_t regctr{
65    ISM(TIMER0_vect){
66        if(attempts == 0xC){
67            icounter+=
68                if((counter & 0x10) == 0)
69                    PORTD |= (1<<PD0);
70                }
71        if((regctr == 0x00)&&(regctr == 0x10))
72            if((regctr == 0x00)&&(regctr == 0x10))
73                if((regctr == 0x00)&&(regctr == 0x10))
74                    if((regctr == 0x00)&&(regctr == 0x10))
75                        if((regctr == 0x00)&&(regctr == 0x10));
76                    else if((regctr == 0x00))
77                        send(receivechar[2]);
78                    else if((regctr == 0x10))
79                        send(receivechar[1]);
80                    else if((regctr == 0x20))
81                        send(receivechar[3]);
82                }
83        send((regctr));
84        latch();
85    if((regctr == 0x00))

```

Figure 6: Raw Code Pt.1

```

83        if((regctr == 0x00))
84            latch();
85    if((regctr == 0x00))

```

Figure 7: Raw Code Pt.2

```

// workspace.v8 - Final Project Raw Code/main.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
[main.c] [main.c] [main.c]
88     sregtr = 0x88;
89     sregtc = sregtr<<2;
90 }
91 // 0 = 0xF0 5 = 0x80
92 // 1 = 0xE0 6 = 0x90
93 // 2 = 0xD0 7 = 0xA0
94 // 3 = 0xC0 8 = 0xB0
95 // 4 = 0x80 9 = 0xC0
96 void debounce(void){
97     uint8_t t;
98     for(uint8_t i=0;i<2;j+=2)
99         for(uint8_t k=0;k<2;j+=2)
100            asm("nop");
101 }
102 void tone(void){
103     PORTD |= (1<<PD0);
104     for(uint8_t i=0;i<2;j+=2)
105         for(uint8_t k=0;k<2;j+=2)
106             asm("nop");
107     PORTD |= (1<<PD1);
108     for(uint8_t i=0;i<2;j+=2)
109         for(uint8_t k=0;k<2;j+=2)
110             asm("nop");
111     PORTD |= (1<<PD2);
112     for(uint8_t i=0;i<2;j+=2)
113         for(uint8_t k=0;k<2;j+=2)
114             asm("nop");
115     PORTD |= (1<<PD3);
116     for(uint8_t i=0;i<2;j+=2)
117         for(uint8_t k=0;k<2;j+=2)
118             for(uint8_t l=0;l<2;j+=2)
119                 for(uint8_t m=0;m<2;j+=2)
120                     for(uint8_t n=0;n<2;j+=2)
121                         asm("nop");
122     PORTD &= ~(1<<PD0);
123     for(uint8_t i=0;i<2;j+=2)
124         for(uint8_t k=0;k<2;j+=2)
125             asm("nop");
126 }
127 void twotone(uint8_t num1, uint8_t delay1,uint8_t num2, uint8_t delay2){
128     for(uint8_t i=0;i<2;j+=2){
129         for(uint8_t k=0;k<2;j+=2)
130             for(uint8_t l=0;l<2;j+=2)
131                 for(uint8_t m=0;m<2;j+=2)
132                     for(uint8_t n=0;n<2;j+=2)
133                         for(uint8_t o=0;o<2;j+=2)
134                             for(uint8_t p=0;p<2;j+=2)
135                                 for(uint8_t q=0;q<2;j+=2)
136                                     for(uint8_t r=0;r<2;j+=2)
137                                         for(uint8_t s=0;s<2;j+=2)
138                                             for(uint8_t t=0;t<2;j+=2)
139                                                 for(uint8_t u=0;u<2;j+=2)
140                                                     for(uint8_t v=0;v<2;j+=2)
141                                                         for(uint8_t w=0;w<2;j+=2)
142                                                             for(uint8_t x=0;x<2;j+=2)
143                                                               for(uint8_t y=0;y<2;j+=2)
144                     PORTB = 0xF0;
145                     PORTD = 0x00;
146                     PORTD |= 0xF0;
147                     PORTD |= (1<<PD5);
148                     //button_int init
149                     TCCR0A = (1<<CS00);
150                     TCCR0B = (1<<OC1TOB);
151                     sei();
152                     sei();
153                     while(1){
154                         if(attemp1 != 0xF0)
155                             PORTB = ~0x00;
156                         if(PORTB & (1<<PB0)) == (0x00) {
157                             tone();
158                             receivedchar[keypadctr] = 0x00;
159                         }
160                         receivedchar[keypadctr] += 0x00;
161                         while(PORTB & (1<<PB1)) != (1<<PB1);
162                         keypadctr++;
163                     }
164                     else if(PORTB & (1<<PB2)) == (0x00) {
165                         tone();
166                         receivedchar[keypadctr] = 0x00;
167                         debounce();
168                         receivedchar[keypadctr] += 0x00;
169                         while(PORTB & (1<<PB2)) != (1<<PB2);
170                         keypadctr++;
171                     }
172                     else if(PORTB & (1<<PB3)) == (0x00) {
173                         tone();
174                         receivedchar[keypadctr] = 0x00;
175                         debounce();
176                         receivedchar[keypadctr] += 0x00;
177                         while(PORTB & (1<<PB3)) != (1<<PB3);
178                         keypadctr++;
179                     }
180                     else if(PORTB & (1<<PB4)) == (0x00) {
181                         tone();
182                         receivedchar[keypadctr] = 0x00;
183                         debounce();
184                         receivedchar[keypadctr] += 0x00;
185                         while(PORTB & (1<<PB4)) != (1<<PB4);
186                         keypadctr++;
187                     }
188                     else if(PORTB & (1<<PB5)) == (0x00) {
189                         tone();
190                         receivedchar[keypadctr] = 0x00;
191                         debounce();
192                         receivedchar[keypadctr] += 0x00;
193                         while(PORTB & (1<<PB5)) != (1<<PB5);
194                         keypadctr++;
195                     }
196                     else if(PORTB & (1<<PB6)) == (0x00) {
197                         tone();
198                         receivedchar[keypadctr] = 0x00;
199                         debounce();
200                         receivedchar[keypadctr] += 0x00;
201                         while(PORTB & (1<<PB6)) != (1<<PB6);
202                         keypadctr++;
203                     }
204                     else if(PORTB & (1<<PB7)) == (0x00) {
205                         tone();
206                         receivedchar[keypadctr] = 0x00;
207                         debounce();
208                         receivedchar[keypadctr] += 0x00;
209                         while(PORTB & (1<<PB7)) != (1<<PB7);
210                         keypadctr++;
211                     }
212                     else if(PORTB & (1<<PB8)) == (0x00) {
213                         tone();
214                         receivedchar[keypadctr] = 0x00;
215                         debounce();
216                         receivedchar[keypadctr] += 0x00;
217                         while(PORTB & (1<<PB8)) != (1<<PB8);
218                         keypadctr++;
219                     }
220                     else if(PORTB & (1<<PB9)) == (0x00) {
221                         tone();
222                         receivedchar[keypadctr] = 0x00;
223                         debounce();
224                         receivedchar[keypadctr] += 0x00;
225                         while(PORTB & (1<<PB9)) != (1<<PB9);
226                         keypadctr++;
227                     }
228                     else if(PORTB & (1<<PB10)) == (0x00) {
229                         tone();
230                         receivedchar[keypadctr] = 0x00;
231                         debounce();
232                         receivedchar[keypadctr] += 0x00;
233                         while(PORTB & (1<<PB10)) != (1<<PB10);
234                         keypadctr++;
235                     }
236                     else if(PORTB & (1<<PB11)) == (0x00) {
237                         tone();
238                         receivedchar[keypadctr] = 0x00;
239                         debounce();
240                         receivedchar[keypadctr] += 0x00;
241                         while(PORTB & (1<<PB11)) != (1<<PB11);
242                         keypadctr++;
243                     }
244                     else if(PORTB & (1<<PB12)) == (0x00) {
245                         tone();
246                         receivedchar[keypadctr] = 0x00;
247                         debounce();
248                         receivedchar[keypadctr] += 0x00;
249                         while(PORTB & (1<<PB12)) != (1<<PB12);
250                         keypadctr++;
251                     }
252                     else if(PORTB & (1<<PB13)) == (0x00) {
253                         tone();
254                         receivedchar[keypadctr] = 0x00;
255                         debounce();
256                         receivedchar[keypadctr] += 0x00;
257                         while(PORTB & (1<<PB13)) != (1<<PB13);
258                         keypadctr++;
259                     }
260                     else if(PORTB & (1<<PB14)) == (0x00) {
261                         tone();
262                         receivedchar[keypadctr] = 0x00;
263                         debounce();
264                         receivedchar[keypadctr] += 0x00;
265                         while(PORTB & (1<<PB14)) != (1<<PB14);
266                         keypadctr++;
267                     }
268                     else if(PORTB & (1<<PB15)) == (0x00) {
269                         tone();
270                         receivedchar[keypadctr] = 0x00;
271                         debounce();
272                         receivedchar[keypadctr] += 0x00;
273                         while(PORTB & (1<<PB15)) != (1<<PB15);
274                         keypadctr++;
275                     }
276                     else if(PORTB & (1<<PB16)) == (0x00) {
277                         tone();
278                         receivedchar[keypadctr] = 0x00;
279                         debounce();
280                         receivedchar[keypadctr] += 0x00;
281                         while(PORTB & (1<<PB16)) != (1<<PB16);
282                         keypadctr++;
283                     }
284                     else if(PORTB & (1<<PB17)) == (0x00) {
285                         tone();
286                         receivedchar[keypadctr] = 0x00;
287                         debounce();
288                         receivedchar[keypadctr] += 0x00;
289                         while(PORTB & (1<<PB17)) != (1<<PB17);
290                         keypadctr++;
291                     }
292                     else if(PORTB & (1<<PB18)) == (0x00) {
293                         tone();
294                         receivedchar[keypadctr] = 0x00;
295                         debounce();
296                         receivedchar[keypadctr] += 0x00;
297                         while(PORTB & (1<<PB18)) != (1<<PB18);
298                         keypadctr++;
299                     }
299 }

```

Figure 8: Raw Code Pt.3

```

// workspace.v8 - Final Project Raw Code/main.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
[main.c] [main.c] [main.c]
129 }
130 int main(void{
131     DDRB = 0x00;
132     DDRD = 0xFF;
133     DDRD |= 0xF0;
134     //button_int init
135     TCCR0A = 0;
136     TCCR0B = (1<<OC1TOB);
137     sei();
138     sei();
139     while(1){
140         if(attemp1 != 0xF0)
141             PORTB = ~0x00;
142         if(PORTB & (1<<PB0)) == (0x00) {
143             tone();
144             receivedchar[keypadctr] = 0x00;
145         }
146         receivedchar[keypadctr] += 0x00;
147         while(PORTB & (1<<PB1)) != (1<<PB1);
148         keypadctr++;
149     }
150     else if(PORTB & (1<<PB2)) == (0x00) {
151         tone();
152         receivedchar[keypadctr] = 0x00;
153         debounce();
154         receivedchar[keypadctr] += 0x00;
155         while(PORTB & (1<<PB2)) != (1<<PB2);
156         keypadctr++;
157     }
158     else if(PORTB & (1<<PB3)) == (0x00) {
159         tone();
160         receivedchar[keypadctr] = 0x00;
161         debounce();
162         receivedchar[keypadctr] += 0x00;
163         while(PORTB & (1<<PB3)) != (1<<PB3);
164         keypadctr++;
165     }
166     else if(PORTB & (1<<PB4)) == (0x00) {
167         tone();
168         receivedchar[keypadctr] = 0x00;
169         debounce();
170         receivedchar[keypadctr] += 0x00;
171         while(PORTB & (1<<PB4)) != (1<<PB4);
172         keypadctr++;
173     }
174     else if(PORTB & (1<<PB5)) == (0x00) {
175         tone();
176         receivedchar[keypadctr] = 0x00;
177         debounce();
178         receivedchar[keypadctr] += 0x00;
179         while(PORTB & (1<<PB5)) != (1<<PB5);
180         keypadctr++;
181     }
182     else if(PORTB & (1<<PB6)) == (0x00) {
183         tone();
184         receivedchar[keypadctr] = 0x00;
185         debounce();
186         receivedchar[keypadctr] += 0x00;
187         while(PORTB & (1<<PB6)) != (1<<PB6);
188         keypadctr++;
189     }
190     else if(PORTB & (1<<PB7)) == (0x00) {
191         tone();
192         receivedchar[keypadctr] = 0x00;
193         debounce();
194         receivedchar[keypadctr] += 0x00;
195         while(PORTB & (1<<PB7)) != (1<<PB7);
196         keypadctr++;
197     }
198     else if(PORTB & (1<<PB8)) == (0x00) {
199         tone();
200         receivedchar[keypadctr] = 0x00;
201         debounce();
202         receivedchar[keypadctr] += 0x00;
203         while(PORTB & (1<<PB8)) != (1<<PB8);
204         keypadctr++;
205     }
206     else if(PORTB & (1<<PB9)) == (0x00) {
207         tone();
208         receivedchar[keypadctr] = 0x00;
209         debounce();
210         receivedchar[keypadctr] += 0x00;
211         while(PORTB & (1<<PB9)) != (1<<PB9);
212         keypadctr++;
213     }
214     else if(PORTB & (1<<PB10)) == (0x00) {
215         tone();
216         receivedchar[keypadctr] = 0x00;
217         debounce();
218         receivedchar[keypadctr] += 0x00;
219         while(PORTB & (1<<PB10)) != (1<<PB10);
220         keypadctr++;
221     }
222     else if(PORTB & (1<<PB11)) == (0x00) {
223         tone();
224         receivedchar[keypadctr] = 0x00;
225         debounce();
226         receivedchar[keypadctr] += 0x00;
227         while(PORTB & (1<<PB11)) != (1<<PB11);
228         keypadctr++;
229     }
223 }

```

Figure 9: Raw Code Pt.4

```

168     receivedchars[keypadctr] = 0x21;/3
169     while((P10B & (1<<P03)) != (1<<P03));
170         keypadctr++;
171     }
172     else if((P10B & (1<<P04)) == (0x00)){
173         tone();
174         debounce();
175         receivedchars[keypadctr] = 0x65;/4
176         while((P10B & (1<<P04)) != (1<<P04));
177         keypadctr++;
178     }
179     else if((P10B & (1<<P05)) == (0x00)){
180         tone();
181         debounce();
182         receivedchars[keypadctr] = 0x86;/5
183         while((P10B & (1<<P05)) != (1<<P05));
184         keypadctr++;
185     }
186     else if((P10B & (1<<P06)) == (0x00)){
187         tone();
188         debounce();
189         receivedchars[keypadctr] = 0x41;/6
190         while((P10B & (1<<P06)) != (1<<P06));
191         keypadctr++;
192     }
193     else if((P10B & (1<<P07)) == (0x00)){
194         tone();
195         debounce();
196         receivedchars[keypadctr] = 0x5C;/7
197         while((P10B & (1<<P07)) != (1<<P07));
198         keypadctr++;
199     }
200     else if((P10B & (1<<P00)) == (0x00)){
201         tone();
202         debounce();
203         receivedchars[keypadctr] = 0x41;/8
204         while((P10B & (1<<P00)) != (1<<P00));
205         keypadctr++;
206     }
207     else if((P10B & (1<<P01)) == (0x00)){
208         tone();
209         debounce();
210         receivedchars[keypadctr] = 0x6F;/9
211         while((P10B & (1<<P01)) != (1<<P01));
212         keypadctr++;
213     }
214     else if((P10B & (1<<P02)) == (0x00)){
215         tone();
216         debounce();
217         receivedchars[keypadctr] = 0x87;/A
218         while((P10B & (1<<P02)) != (1<<P02));
219         keypadctr++;
220     }
221     else if((P10B & (1<<P03)) == (0x00)){
222         tone();
223         debounce();
224         receivedchars[keypadctr] = 0x3E;/B
225         while((P10B & (1<<P03)) != (1<<P03));
226         keypadctr++;
227     }
228     if(keypadctr == 4){
229         for(uint_t i=0;i<5;i++)
230             keypad[i] = receivedchars[i];
231         keypad[5] = 0;
232         if((receivedchars[0] == code[0]) & (receivedchars[1] == code[1]) & (receivedchars[2] == code[2]) & (receivedchars[3] == code[3])){
233             if((PORTD | (1<<P04));
234                 PORTD |= (1<<P03);
235                 PORTD |= (1<<P05);
236                 PORTD |= (1<<P06);
237                 counter = 0;
238                 attempts = 0;
239                 receivedchar[0] = 0x0F;
240                 receivedchar[1] = 0x0E;
241                 receivedchar[2] = 0x0D;
242                 receivedchar[3] = 0x0C;
243                 buzztone(50,12,65,5);
244                 for(uint_t i=0;i<5;i++)
245                     for(uint_t j=0;j<5;j++){
246                         if(j!=i)
247                             tone();
248                         if(attempts == 0x60)
249                             attempts = 0x21;
250                         else if(attempts == 0x21)
251                             attempts = 0x00;
252                     }
253                 counter = 0;
254             }
255         }

```

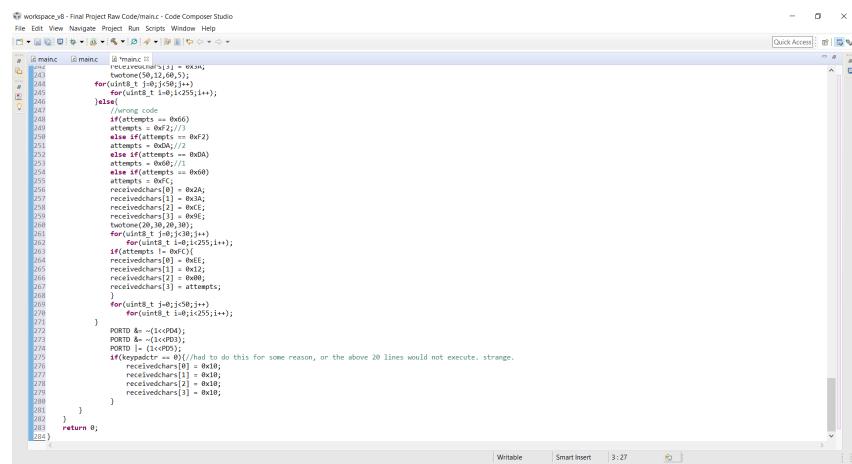
Figure 10: Raw Code Pt.5

```

218     receivedchars[keypadctr] = 0x21;/9
219     while((P11B & (1<<P11)) != (1<<P11));
220         keypadctr++;
221     }
222     else if((P11B & (1<<P12)) == (0x00)){
223         tone();
224         debounce();
225         receivedchars[keypadctr] = 0x41;/A
226         while((P11B & (1<<P12)) != (1<<P12));
227         keypadctr++;
228     }
229     else if((P11B & (1<<P13)) == (0x00)){
230         tone();
231         debounce();
232         receivedchars[keypadctr] = 0x6F;/B
233         while((P11B & (1<<P13)) != (1<<P13));
234         keypadctr++;
235     }
236     if(keypadctr == 4){
237         for(uint_t i=0;i<5;i++)
238             keypad[i] = receivedchars[i];
239         keypad[5] = 0;
240         if((receivedchars[0] == code[0]) & (receivedchars[1] == code[1]) & (receivedchars[2] == code[2]) & (receivedchars[3] == code[3])){
241             if((PORTD | (1<<P04));
242                 PORTD |= (1<<P03);
243                 PORTD |= (1<<P05);
244                 PORTD |= (1<<P06);
245                 counter = 0;
246                 attempts = 0;
247                 receivedchar[0] = 0x0F;
248                 receivedchar[1] = 0x0E;
249                 receivedchar[2] = 0x0D;
250                 receivedchar[3] = 0x0C;
251                 buzztone(50,12,65,5);
252                 for(uint_t i=0;i<5;i++)
253                     for(uint_t j=0;j<5;j++){
254                         if(j!=i)
255                             tone();
256                         if(attempts == 0x60)
257                             attempts = 0x21;
258                         else if(attempts == 0x21)
259                             attempts = 0x00;
260                     }
261                 counter = 0;
262             }
263         }

```

Figure 11: Raw Code Pt.6



```

// workspace.c8 - Final Project Raw Code (main.c - Code Composer Studio)
File Edit View Navigate Project Run Songs Window Help
[main.c] [main.c]
243     receivedchar[i] = exm;
244     txchar(59,12,0x57);
245     for(uint8_t i=0;i<4;i++);
246     for(uint8_t i=0;i<255;i++);
247     } // wrong code
248     if(attempts == 0x00)
249     attempts = 0x01; // 1
250     else if(attempts == 0x02)
251     attempts = 0x0A; // 10
252     else if(attempts == 0x04)
253     attempts = 0x0B; // 11
254     else if(attempts == 0x08)
255     attempts = 0x09;
256     receivedchar[0] = 0x2A;
257     receivedchar[1] = 0x24;
258     receivedchar[2] = 0x41;
259     receivedchar[3] = 0x42;
260     txchar(59,12,0x57);
261     for(uint8_t i=0;i<30;i++);
262     for(uint8_t i=0;i<255;i++);
263     if(attempts == 0x0C)
264     receivedchar[0] = 0x41;
265     receivedchar[1] = 0x42;
266     receivedchar[2] = 0x40;
267     receivedchar[3] = attempts;
268     for(uint8_t i=0;i<30;i++);
269     for(uint8_t i=0;i<255;i++);
270     }
271     PORD0 &= ~(1<<POD0);
272     PORD1 |= (1<<POD1);
273     PORD2 |= (1<<POD2);
274     if(keypaddr == 0) // had to do this for some reason, or the above 20 lines would not execute. strange.
275     receivedchar[0] = 0x10;
276     receivedchar[1] = 0x10;
277     receivedchar[2] = 0x10;
278     receivedchar[3] = 0x10;
279     }
280   }
281 }
282 }
283 return 0;
284

```

Figure 12: Raw Code Pt.7