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.

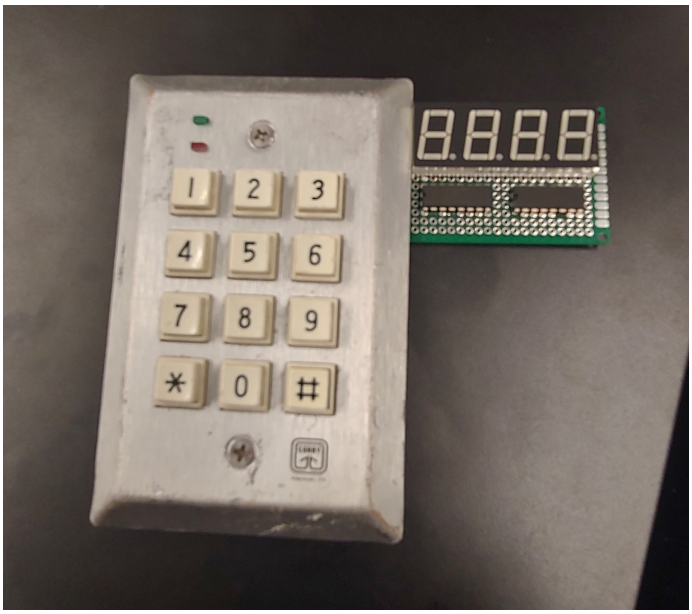


Figure 1: Keypad Components

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.

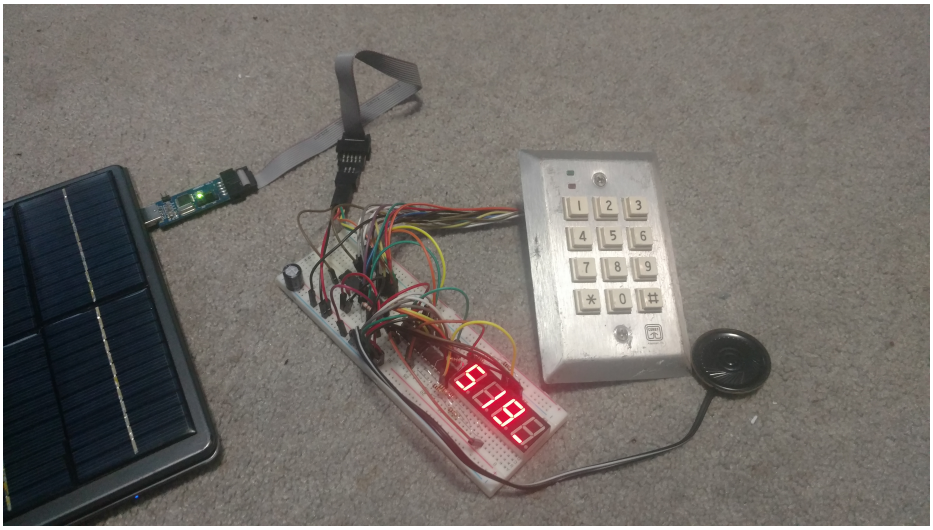


Figure 2: Breadboard Demo

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.

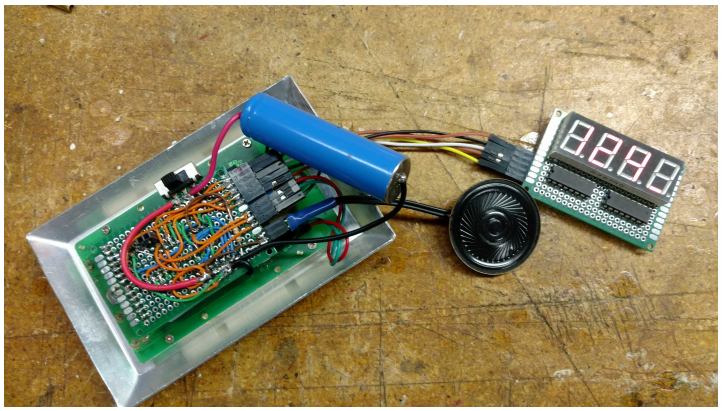


Figure 3: Keypad System with Lithium Battery

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

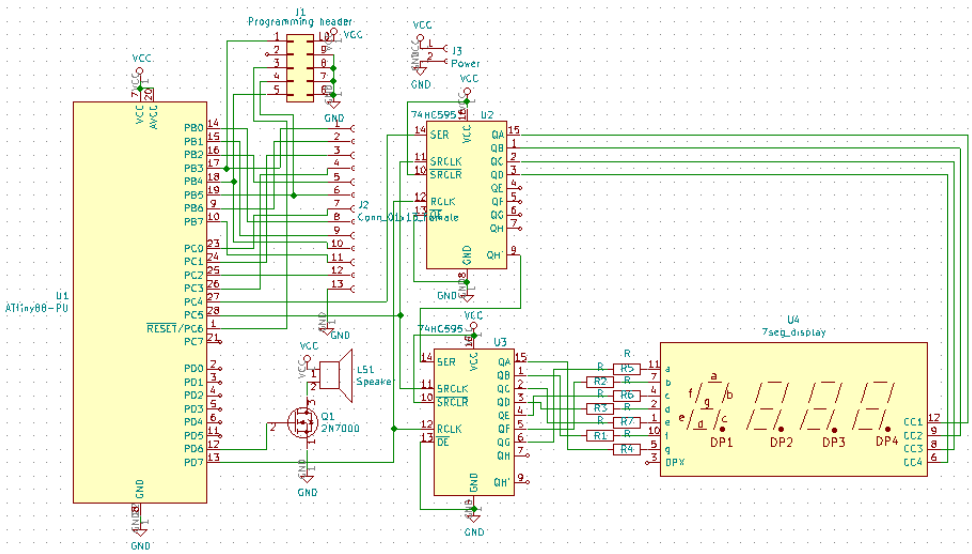


Figure 4: System Schematic Architecture

7.2 Appendix B: Spiral Design

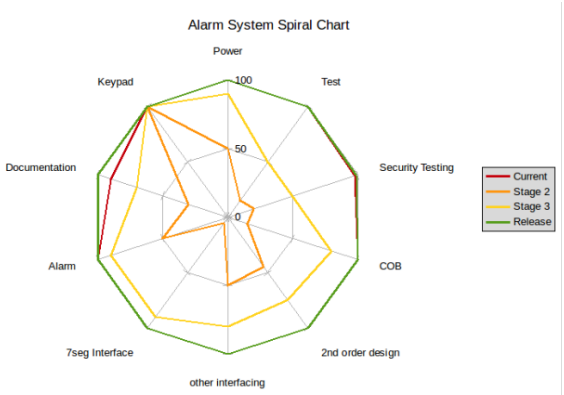


Figure 5: Spiral Design Progress

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

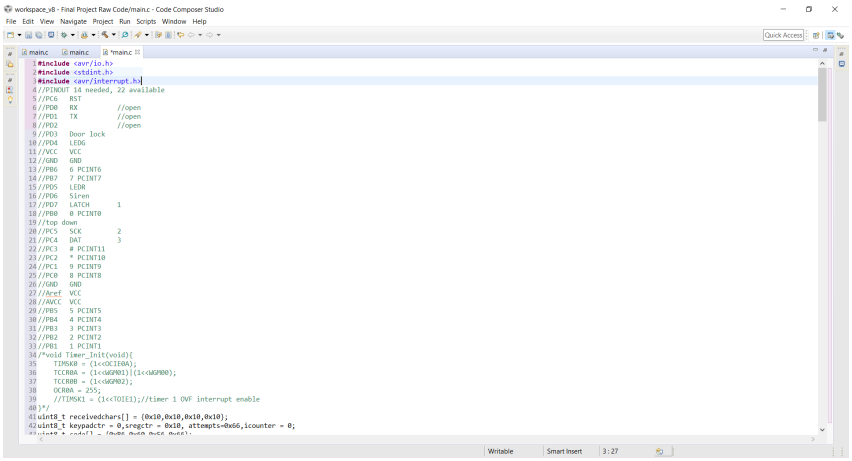


Figure 6: Raw Code Pt.1

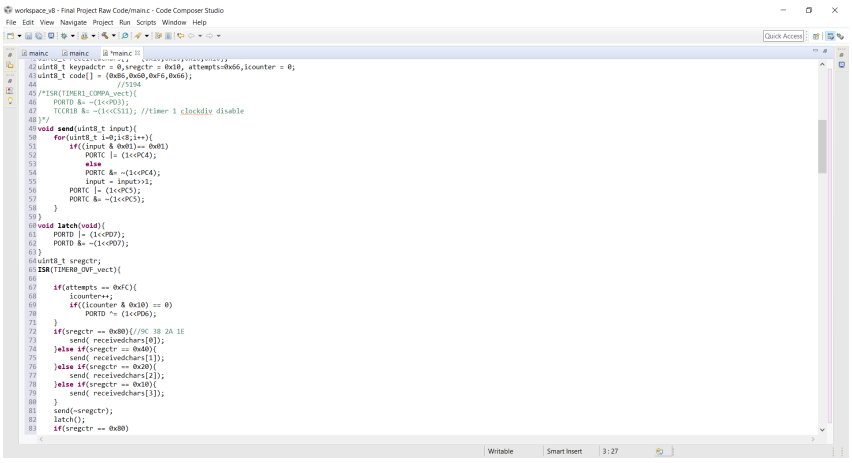
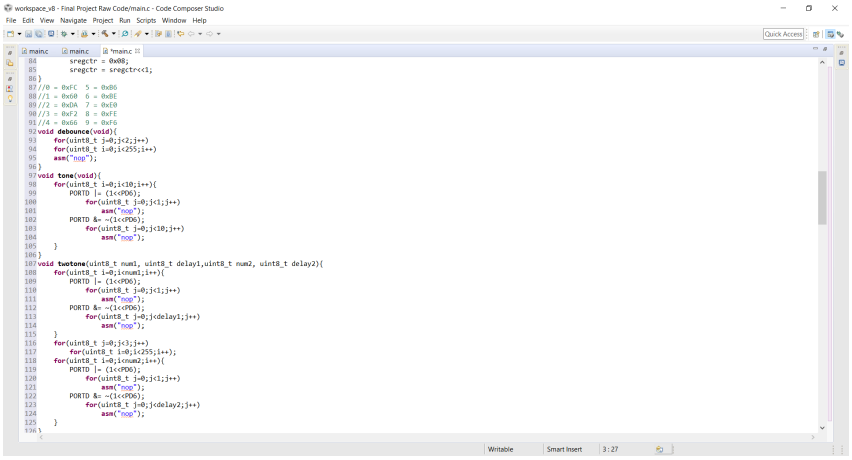
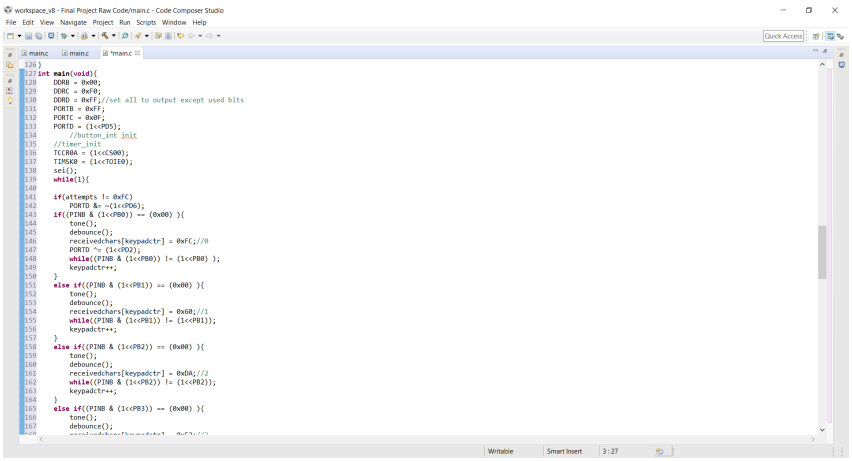


Figure 7: Raw Code Pt.2



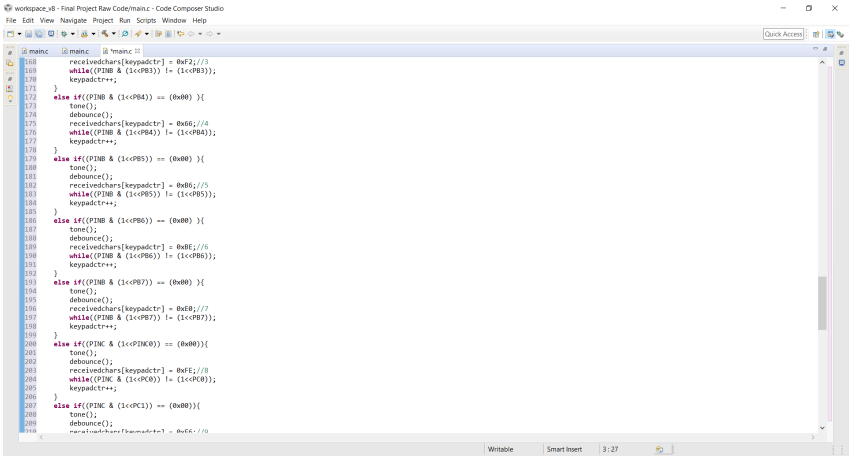
```
88          sregctr = 0x08;
89          sregctr = sregctr<<1;
90      }
91  //0 = 0xFC  5 = 0xB6
92  //1 = 0x60  6 = 0xBE
93  //2 = 0xDA  7 = 0xE0
94  //3 = 0xF2  8 = 0xFE
95  //4 = 0x66  9 = 0xF6
96  void debounce(void){
97      for(uint8_t j=0;j<2;j++)
98      for(uint8_t i=0;i<255;i++)
99      asm("nop");
100 }
101 void tone(void){
102     for(uint8_t i=0;ic10;i++){
103         PORTD |= (1<<PD6);
104             for(uint8_t j=0;j<1;j++)
105                 asm("nop");
106         PORTD &= ~(1<<PD6);
107             for(uint8_t j=0;j<10;j++)
108                 asm("nop");
109     }
110 }
111 void twotone(uint8_t num1, uint8_t delay1,uint8_t num2, uint8_t delay2){
112     for(uint8_t i=0;icnum1;i++){
113         PORTD |= (1<<PD6);
114             for(uint8_t j=0;j<1;j++)
115                 asm("nop");
116         PORTD &= ~(1<<PD6);
117             for(uint8_t j=0;j<delay1;j++)
118                 asm("nop");
119     }
120     for(uint8_t j=0;j<3;j++)
121         for(uint8_t i=0;i<255;i++);
122     for(uint8_t i=0;icnum2;i++){
123         PORTD |= (1<<PD6);
124             for(uint8_t j=0;j<1;j++)
125                 asm("nop");
126         PORTD &= ~(1<<PD6);
127             for(uint8_t j=0;j<delay2;j++)
128                 asm("nop");
129     }
130 }
```

Figure 8: Raw Code Pt.3



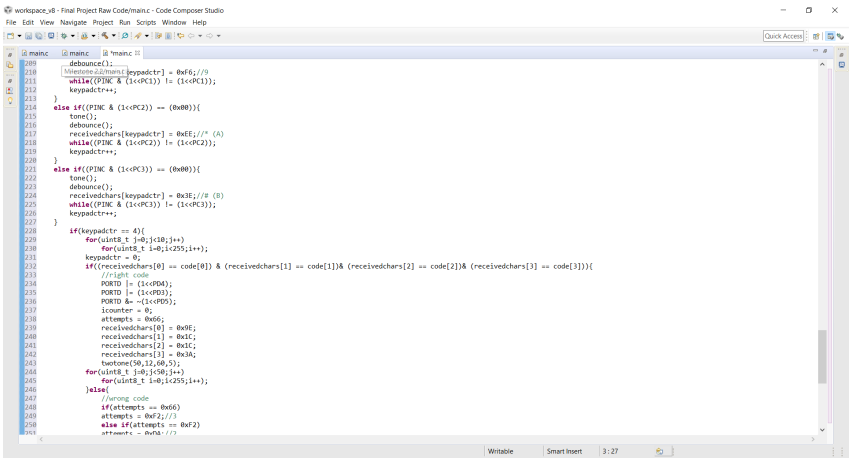
```
126 }
127 int main(void){
128     DDRB = 0x00;
129     DDRC = 0xF0;
130     DDRD = 0xFF;//set all to output except used bits
131     PORTB = 0xFF;
132     PORTC = 0x0F;
133     PORTD = (1<<PD5);
134         //button_int init
135     //timer_init
136     TCCR0A = (1<<CS00);
137     TIMSK0 = (1<<TOIE0);
138     sei();
139     while(1){
140
141     if(attempts != 0xFC)
142         PORTD &= ~(1<<PD6);
143     if((PINB & (1<<PB0)) == (0x00) ){
144         tone();
145         debounce();
146         receivedchars[keypadctr] = 0xFC;//0
147         PORTD ^= (1<<PD2);
148         while((PINB & (1<<PB0)) != (1<<PB0) );
149         keypadctr++;
150     }
151     else if((PINB & (1<<PB1)) == (0x00) ){
152         tone();
153         debounce();
154         receivedchars[keypadctr] = 0x60;//1
155         while((PINB & (1<<PB1)) != (1<<PB1));
156         keypadctr++;
157     }
158     else if((PINB & (1<<PB2)) == (0x00) ){
159         tone();
160         debounce();
161         receivedchars[keypadctr] = 0xDA;//2
162         while((PINB & (1<<PB2)) != (1<<PB2));
163         keypadctr++;
164     }
165     else if((PINB & (1<<PB3)) == (0x00) ){
166         tone();
167         debounce();
```

Figure 9: Raw Code Pt.4



```
        receivedchars[keypadctr] = 0xF2;//3
        while((PINB & (1<<PB3)) != (1<<PB3));
        keypadctr++;
    }
    else if((PINB & (1<<PB4)) == (0x00) ){
        tone();
        debounce();
        receivedchars[keypadctr] = 0x66;//4
        while((PINB & (1<<PB4)) != (1<<PB4));
        keypadctr++;
    }
    else if((PINB & (1<<PB5)) == (0x00) ){
        tone();
        debounce();
        receivedchars[keypadctr] = 0xB6;//5
        while((PINB & (1<<PB5)) != (1<<PB5));
        keypadctr++;
    }
    else if((PINB & (1<<PB6)) == (0x00) ){
        tone();
        debounce();
        receivedchars[keypadctr] = 0xBE;//6
        while((PINB & (1<<PB6)) != (1<<PB6));
        keypadctr++;
    }
    else if((PINB & (1<<PB7)) == (0x00) ){
        tone();
        debounce();
        receivedchars[keypadctr] = 0xE0;//7
        while((PINB & (1<<PB7)) != (1<<PB7));
        keypadctr++;
    }
    else if((PINC & (1<<PINC0)) == (0x00)){
        tone();
        debounce();
        receivedchars[keypadctr] = 0xFE;//8
        while((PINC & (1<<PC0)) != (1<<PC0));
        keypadctr++;
    }
    else if((PINC & (1<<PC1)) == (0x00)){
        tone();
        debounce();
```

Figure 10: Raw Code Pt.5



```
        debounce();
        receivedchars[keypadctr] = 0xF6;//9
        while((PINC & (1<<PC1)) != (1<<PC1));
        keypadctr++;
    }
    else if((PINC & (1<<PC2)) == (0x00)){
        tone();
        debounce();
        receivedchars[keypadctr] = 0xEE;//* (A)
        while((PINC & (1<<PC2)) != (1<<PC2));
        keypadctr++;
    }
    else if((PINC & (1<<PC3)) == (0x00)){
        tone();
        debounce();
        receivedchars[keypadctr] = 0x3E;//# (B)
        while((PINC & (1<<PC3)) != (1<<PC3));
        keypadctr++;
    }
    if(keypadctr == 4){
        for(uint8_t j=0;j<10;j++)
            for(uint8_t i=0;i<255;i++);
        keypadctr = 0;
        if((receivedchars[0] == code[0]) & (receivedchars[1] == code[1])& (receivedchars[2] == code[2])& (receivedchars[3] == code[3])){
            //right code
            PORTD |= (1<<PD4);
            PORTD |= (1<<PD3);
            PORTD &= ~(1<<PD5);
            icounter = 0;
            attempts = 0x66;
            receivedchars[0] = 0x9E;
            receivedchars[1] = 0x1C;
            receivedchars[2] = 0x1C;
            receivedchars[3] = 0x3A;
            twotone(50,12,60,5);
        for(uint8_t j=0;j<50;j++)
            for(uint8_t i=0;i<255;i++);
        }else{
            //wrong code
            if(attempts == 0x66)
            attempts = 0xF2;//3
            else if(attempts == 0xF2)
```

Figure 11: Raw Code Pt.6

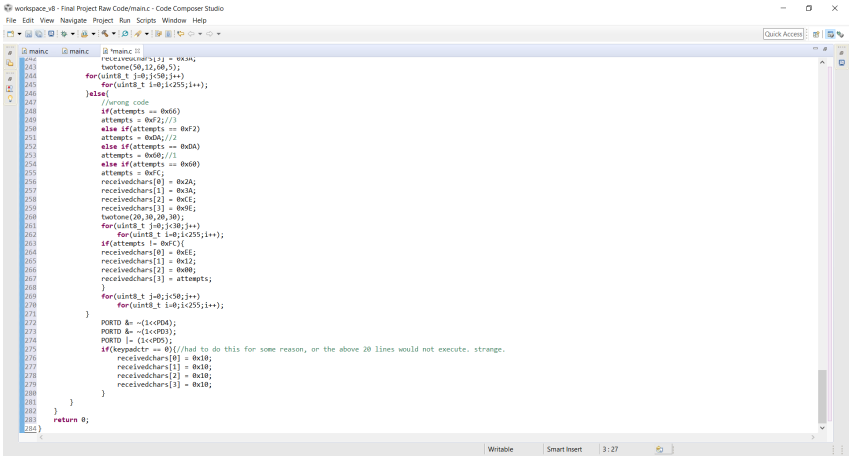


Figure 12: Raw Code Pt.7