NAME: Carolyn Nahikian
DATE: 09/06/2022
UFID: 24535133

EE Design 1 Technical Report
Final Project
"Whack-A-Light"

Introduction

When given the requirements for the final project, I had the idea to design a project that was something useful for after this course is over. I chose to design my own version of the classic carnival game "Whack-A-Mole". I called this game "Whack-A-Light". This design is encased in an aluminum box with a plexiglass cover that contains an LCD display, a speaker, an on/off switch, a volume knob, a mode select button, and 7 LED push buttons.

The goal of this project was to create a device that utilizes push button switches for a timed game that requires the user to push buttons as the LED inside of them turns on. The LCD shows the game mode, which the player selects with the pushbutton switch with the label "MODE". The LCD also displays a count down for the timed game to start, as well as the score for each game, and then finally at the end of each game, the display shows a "GAME OVER!" message and the final score. In addition to these elements, the speaker plays a sound at the start and end of each game over the speaker attached; this sound has controllable volume that is controlled by a knob on the face of the device labeled "VOLUME". With these features, my goal was to create this device with seamless gameplay and smooth audio output.

Methods

For this design, I referenced datasheets for some components I used, including the PIC18F47K40 microcontroller, MPLAB Snap Debugger, LTC1661 DAC, TLV4110 op-amp, and the 7805 voltage regulator. I also referenced my code from past modules I completed earlier in this course.

Analog Input

At least one analog input was required for this project design, using the ADC (Analog to digital converter) to output the analog input to the microprocessor. I chose to use a 10 kOhm potentiometer for my analog input; this was used with the ADC by inputting the voltage output into the ADC and converting it to a 10-bit value between 0 and 1023. A potentiometer works for this analog application because, as a mechanical device, it measures the potential dropped across a segment of wire that carries a constant current that is directly proportional to its length. This then allows a voltage to be used as the analog input to the ADC. With this functionality, I decided to use the potentiometer and a knob to act as the volume control for my analog output, an 8 Ohm speaker.

Analog Output

For my analog output, I used an 8 Ohm speaker to output sine waves of various frequencies using an external DAC (Digital to analog converter) and the internal SPI (serial peripheral interface) of the PIC18F47K40 microprocessor. Using my knowledge from the previous SPI-DAC module from this course, I was able to take a 64-point sine wave and adjust the frequency of it using delays. I also used my analog input and ADC implementation to adjust the gain of the SPI output, thus adjusting the volume of the audio output. This volume control goes across the full range of the ADC, from 0 to 1023, allowing the volume to turn completely off if desired.

Digital Inputs

For this design, I used 8 pushbutton switches as my digital inputs. 7 of the 8 buttons were used for the “moles” in the game, and the last button was used as the mode select. Using multiple functions and many switch cases in my code, I configured these switches to perform the tasks exactly as desired.

Digital Outputs

The digital outputs I chose to use for this design were LEDS. Unlike previous projects in this course, I used momentary illuminated pushbuttons, which work as both a digital input and a digital output. These built-in LEDs were implemented in a way where they lit up randomly to be turned off when the same button was pressed; this implementation allowed for the LEDs to give off the same effect as “moles” in the game of “Whack-A-Mole”.

LCD (Liquid Crystal Display)

Another digital output used for this project is an LCD. I used the LCD in my project to display the game mode, score, and other messages at the start and end of each game session. I was able to implement this by finding some code for a similar LCD’s initialization and adapting it to work for 4-bit data with DB4 through DB7 on the LCD.

PIC Programming

I chose to use the PIC18F47K40 microprocessor for this design because of my previous experience with it from past design modules. This processor has many internal capabilities including ADC, SPI, Timers, and 5 register ports that I used for my implementation.

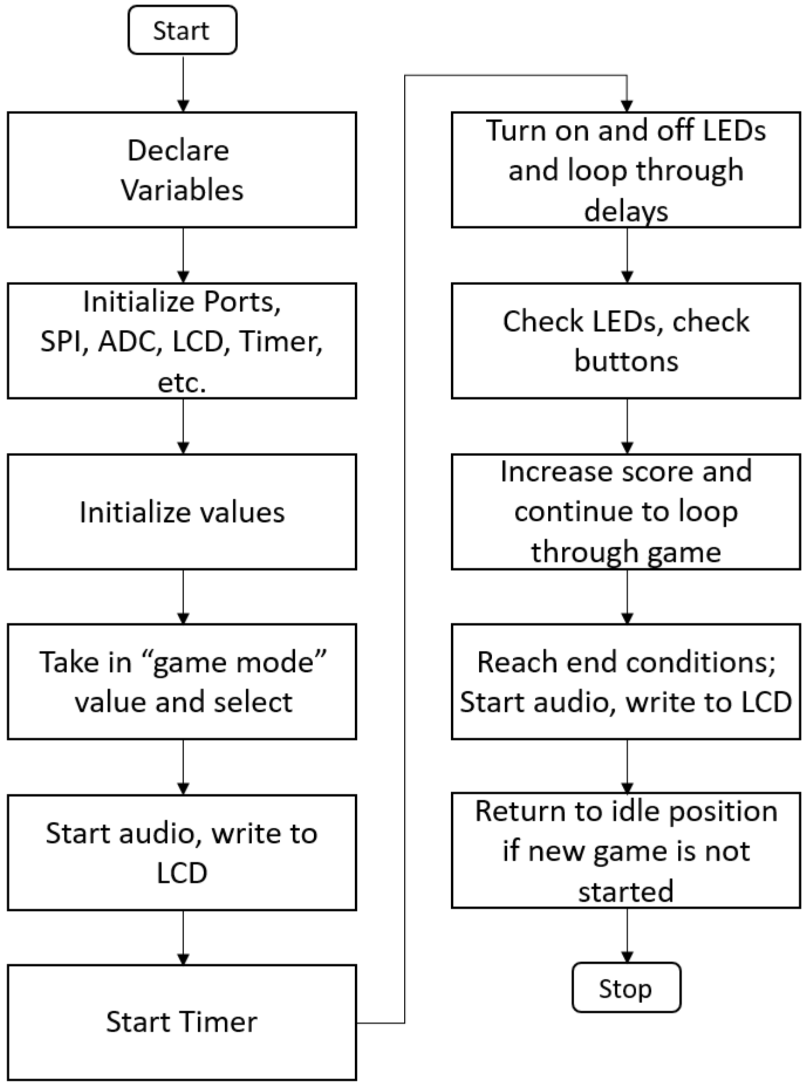
On/Off Switch

For easy control of the power supplied to the device, I added an on/off switch to the outside of the device's case; this allows for the user to turn off the device as you would with any handheld game device.

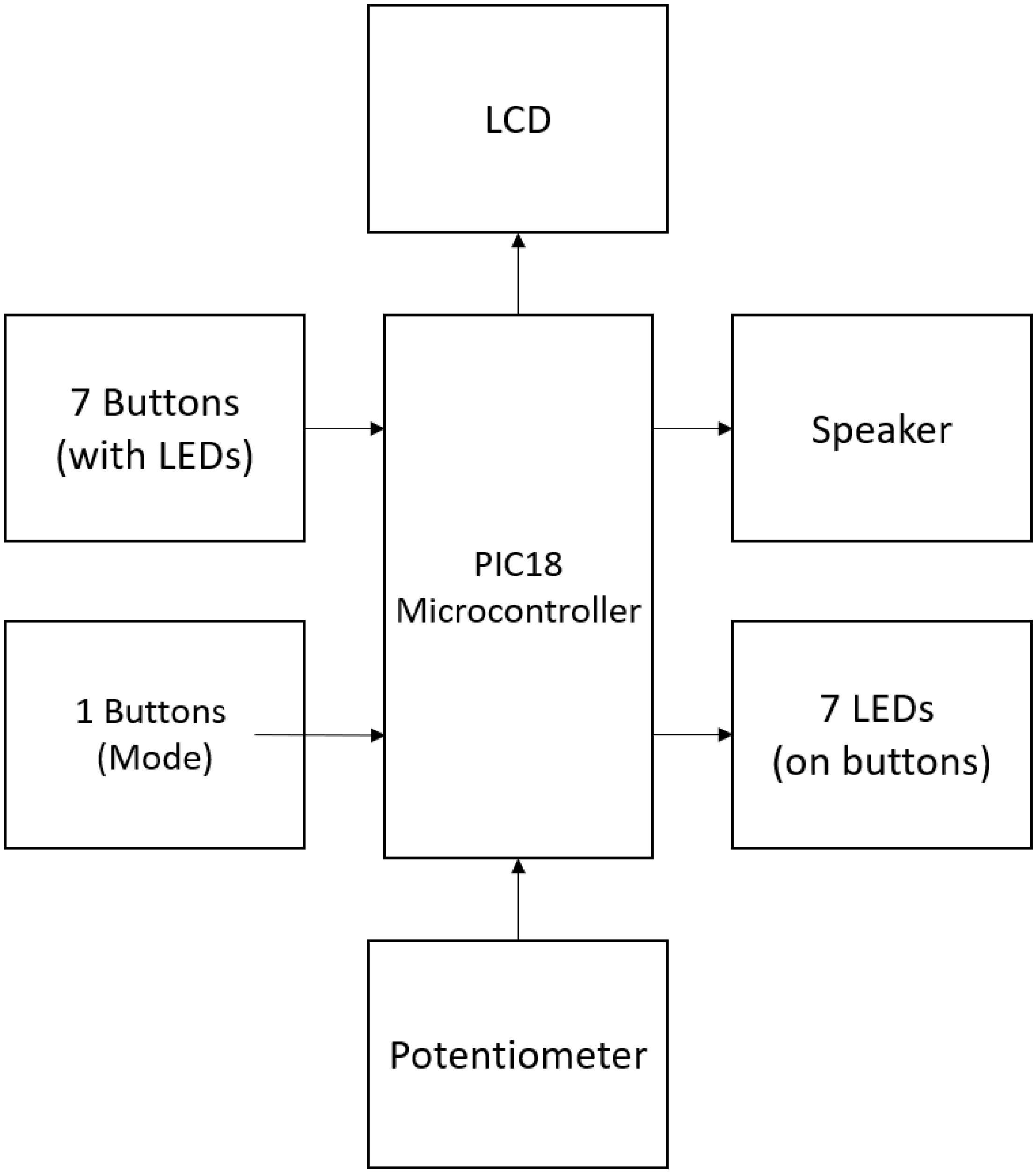
Other Physical Components

In addition to the electronic components in this design, I also used multiple physical components. I used a 6 by 8-inch aluminum box to encase my components and a piece of plexiglass which I cut to the shape of the box to use as a cover. The hold the components in place, I used multiple standoffs, nuts, and screws. I also used rubber feet to put on the bottom of the box so that the metal would not ruin any surfaces that it may lay on. In addition to these components, I also used wires of various types to connect everything together.

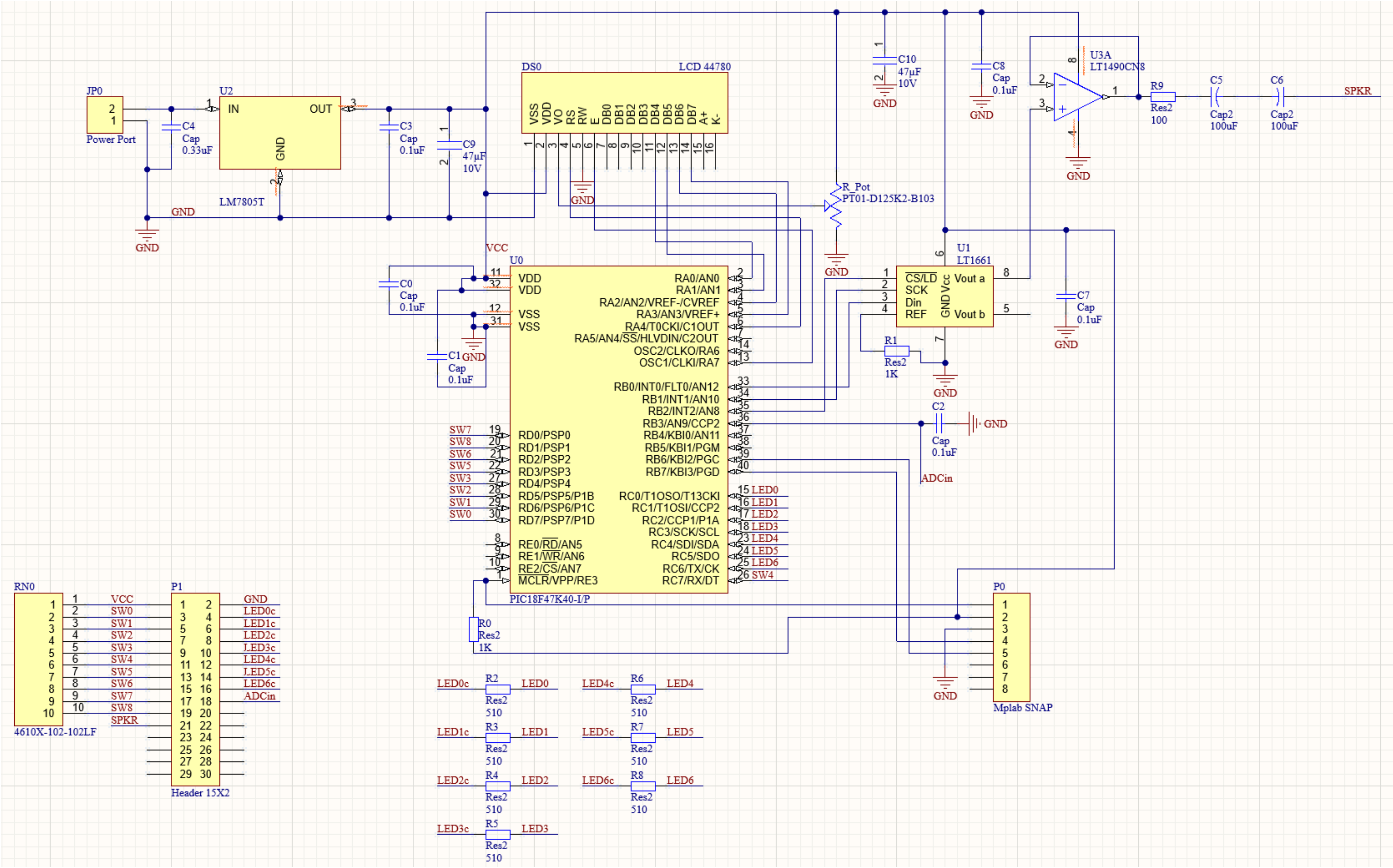
Software Block Diagram


Start
Declare Variables
Initialize Ports, SPI, ADC, LCD, Timer, etc.
Initialize values
Take in "game mode" value and select
Start audio, write to LCD
Start Timer
Turn on and off LEDs and loop through delays
Check LEDs, check buttons
Increase score and continue to loop through game
Reach end conditions; Start audio, write to LCD
Return to idle position if new game is not started
Stop

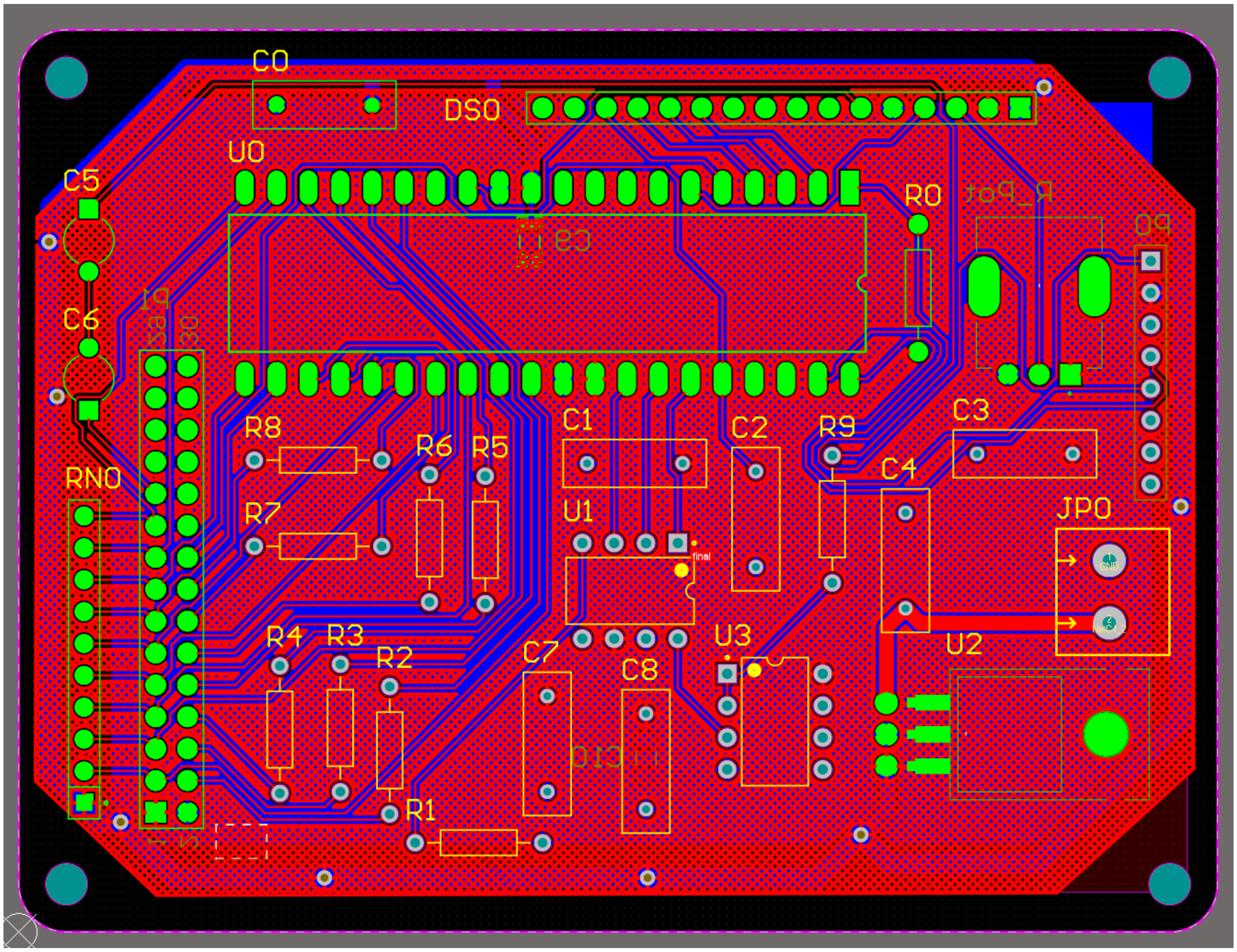
Hardware Block Diagram



Altium Schematic



Altium Board Layout (PCB)



Bill of Materials

Part Number	Number Required	Price for each component
4610X-101-102LF	1	$0.34
L7805CV	1	$0.69
CFR-25JR-52-510R	7	$0.047
ECA-1VM101I	2	$0.36
K104M15X7RF53H5	6	$0.167
K334K20X7RF5TH5	1	$0.84
GRM31CR61A476ME15K	2	$0.254
LTC1661CN8#PBF	1	$5.09
CMS-28528N-L152	1	$3.69
PT01-D115K2-B103	1	$0.79
AC-1413	1	$17.50
P160KNP-0FB15B10K	1	$1.46

GREY 15mm D-SHAFT	1	$1.01
77313-118-30LF	1	$1.38
920-0141-01	1	$7.00
TLV4110IP	1	$2.64
PCBs from Oshpark	3	$53.40 (for 3)
16mm Illuminated Pushbutton – White Momentary	1	$1.50
Amazon Momentary Illuminated Pushbutton	7	$2.00
Plexiglass Sheet	1	$1.75
Velcro Sheets	2	$0.25
Feet for Box	4	$0.20
Standoffs	8	$0.20
Screws and Nuts	20	$0.15 per dozen
Other miscellaneous resistors	1	Included in kit
LCD	1	Included in kit
PIC18F47K40	1	Included in kit
TOTAL:	78	~$117.19

The costs above may be off slightly due to price changes and the number of components actually used.

Results

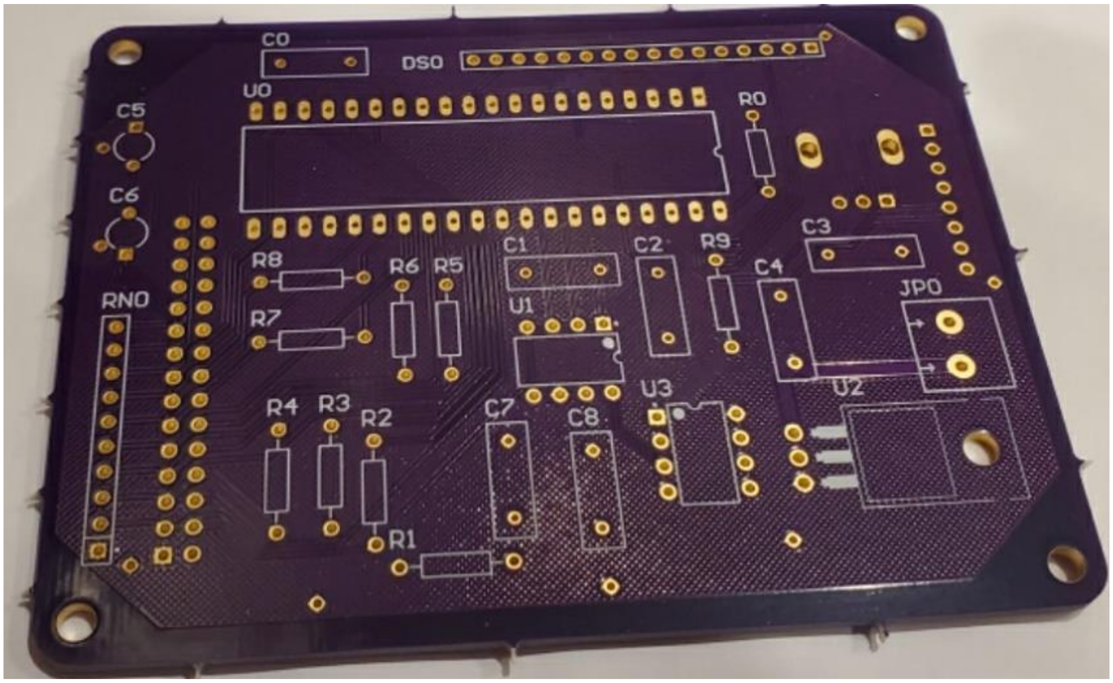


Figure 1 PCB before parts being soldered on

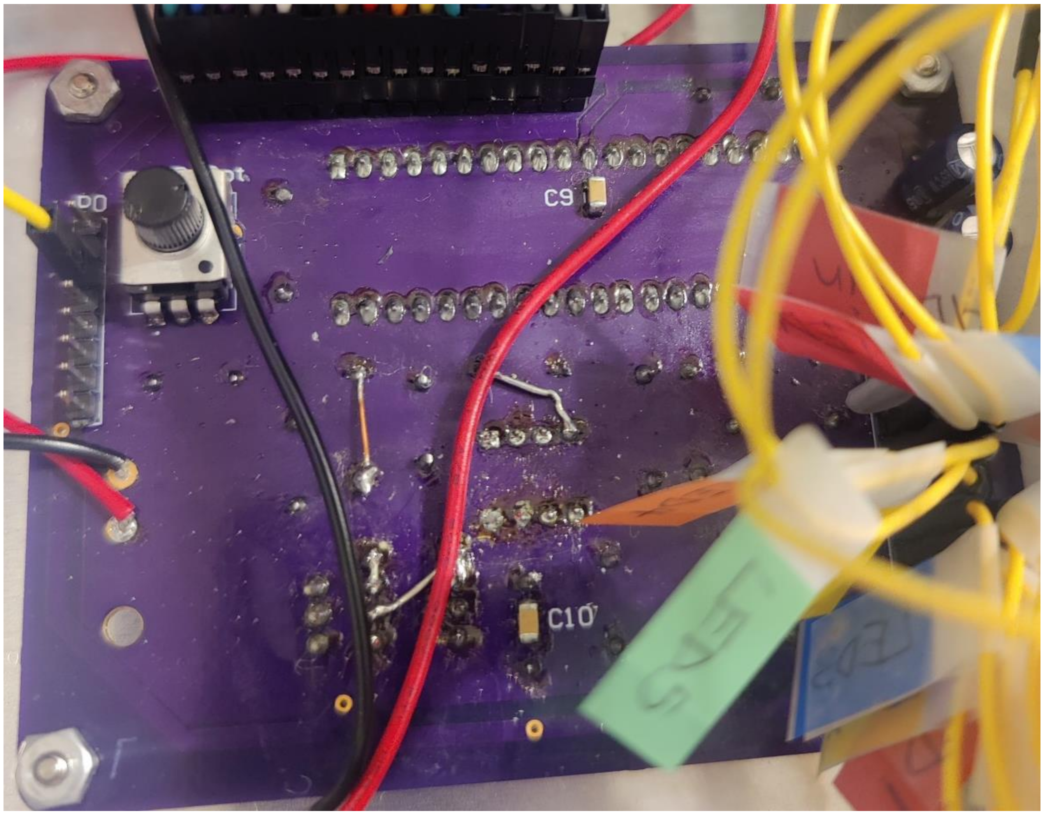


Figure 2 PCB after being soldered and installed in device box

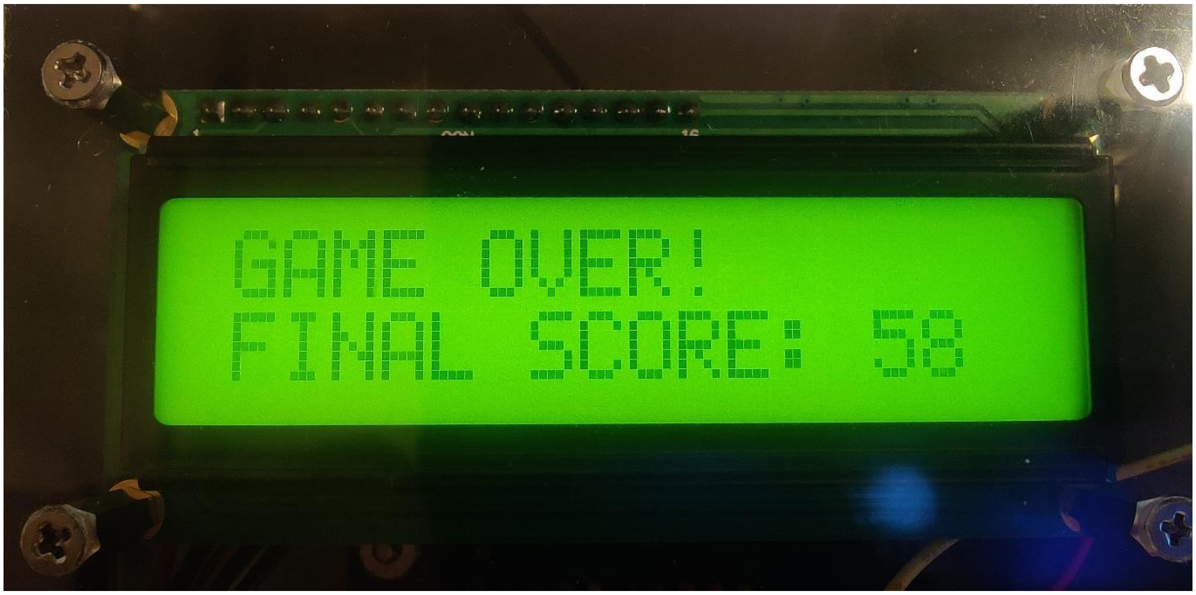


Figure 3 LCD display at end of game on Hard Mode

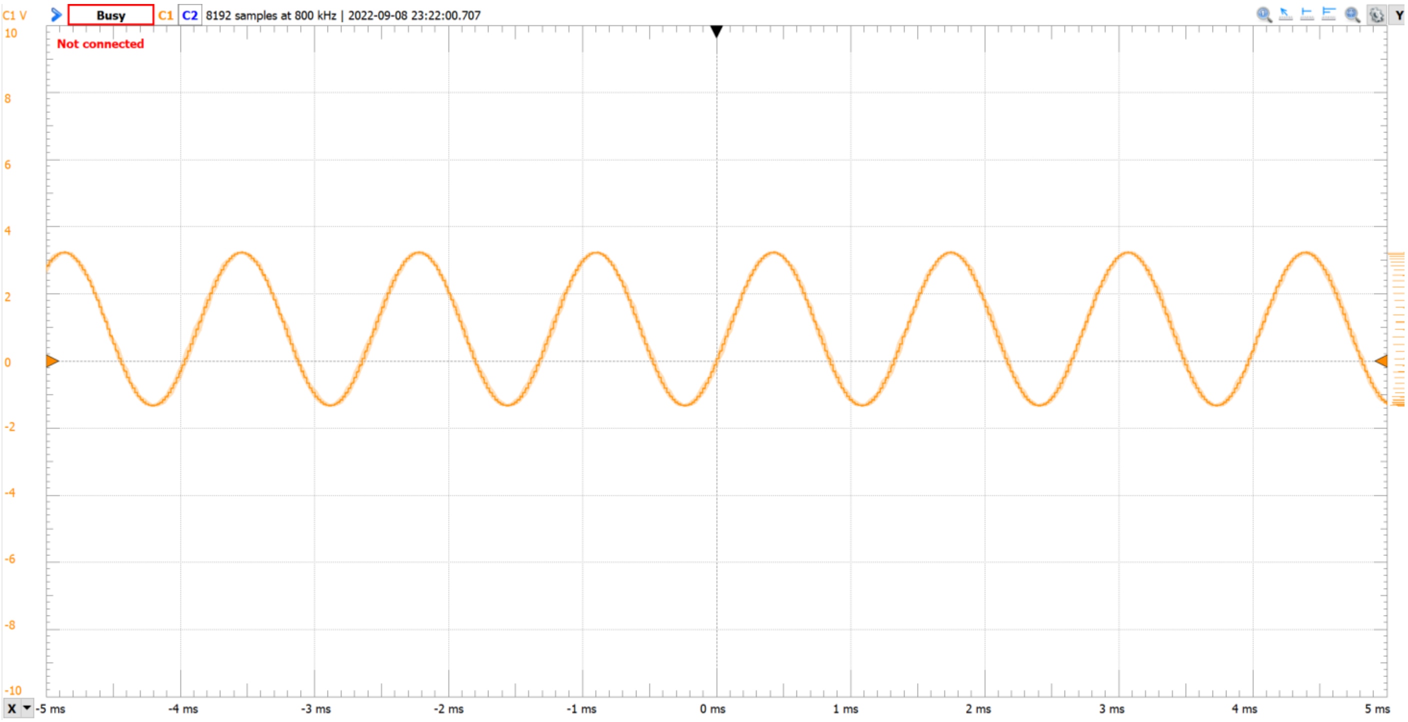


Figure 4 Sine Wave output of speaker during the start of game countdown

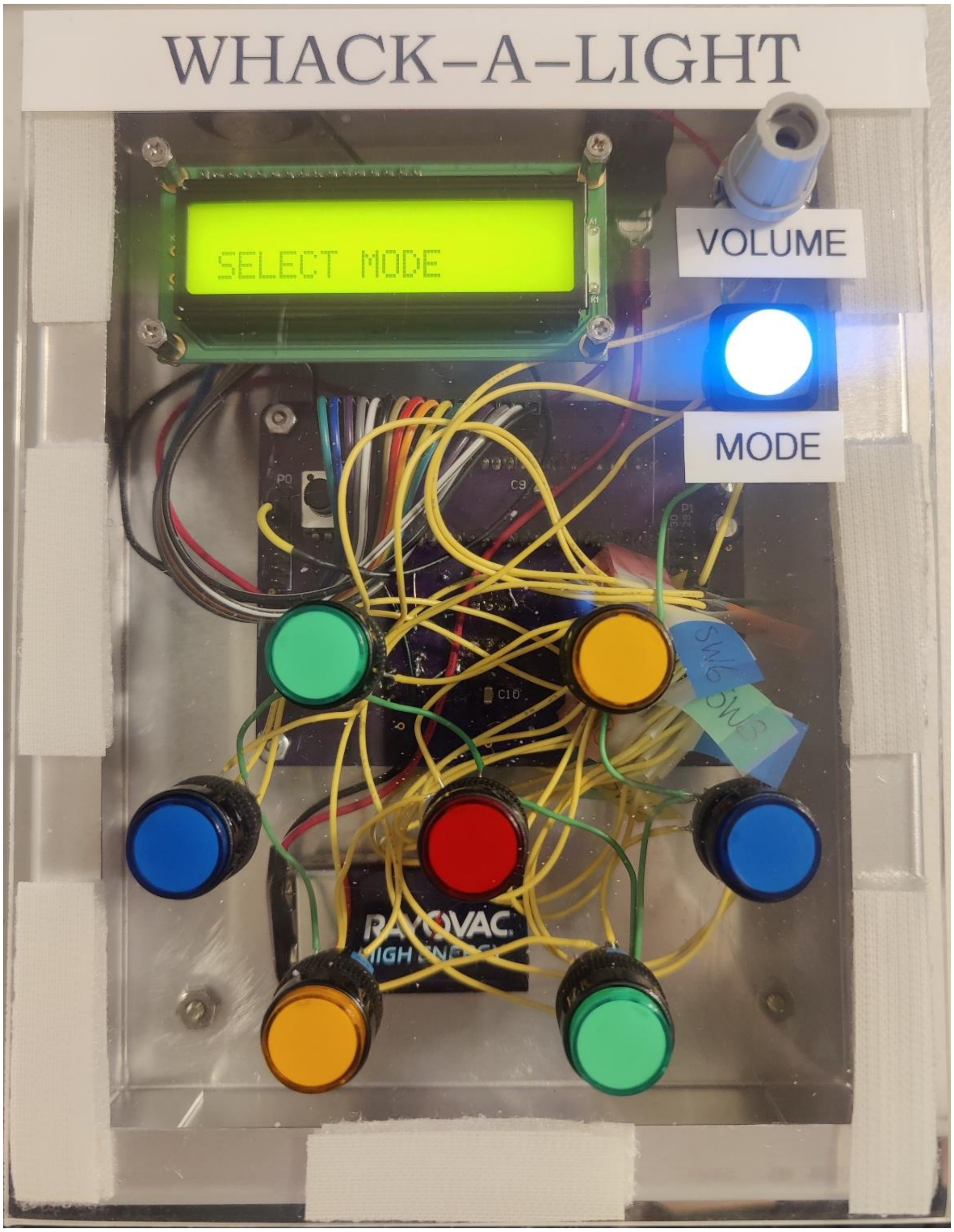


Figure 5 Picture of project fully assembled and enclosed in the aluminum box

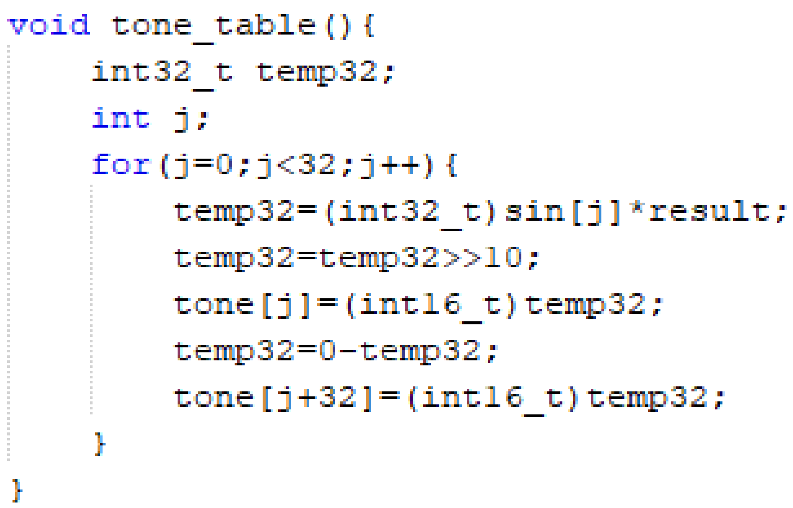
Discussion/Conclusion

Throughout the design process of the project, I ran into many difficulties. The majority of my issues stemmed from the components I chose not meeting the requirements of what I was using them for. An example of this included the op-amp in my design. The op-amp used as an output buffer for the DAC requires a drive current of 312.5 mA with ±2.5V of voltage to drive an 8-Ohm speaker. Originally, I designed my project with a LT1490 op-amp which was provided in our lab kit, however, this op-amp does not reach the needed drive current, so I had to find another op-amp that met the needed requirements and also fit in the footprint of the original design on my PCB. I chose to use the TLV4110 op-amp for this because it drives ±320 mA and it fits the footprint with only 2 wires soldered on.

Another issue I ran into was occurring whenever I tried to program my PIC microcontroller when it was connected to the board. This issue confused me as the PIC would program half of the time and other times it would not. It ended up being a problem with the attached panel and how the LEDs were soldered together; two wires ended up touching after being connected causing the board to short and thus drained the attached battery. Because of this, I also saw issues with the LEDs being very dim and inconsistent. After determining that the battery was the issue, I ended up connecting my design to a 9V transformer that I could plug into the wall while I was debugging.

The last major batch of issues stemmed from the DAC and the analog output through the speaker. First, I noticed that the DAC was incorrectly connected in my PCB, pulling the reference to ground instead of to 5V; this was fixed by removing an unnecessary resistor and attaching a wire to Vcc. Then, I noticed that the output of the op-amp, which goes to the speaker, had a resistor connected to it which was preventing the speaker from producing sound, despite the output showing a sine wave on the oscilloscope. The fix to this issue was connecting a wire across the series resistor.

In addition to the many analog output issues, I ran into during the debugging of this project, I also experienced some issues with the analog input and its implementation with the DAC output to the speaker. After first ensuring that the sine wave output worked properly, I added the ADC implementation by multiplying the sine wave output by the ADC input then dividing it by 1024 to have the output change with the ratio of the potentiometer and knob. This did successfully adjust the volume of the sound, however, with this added math in the original sound function I made, the delay was slowed down too much and ended up producing a sine wave that was a slower frequency at the base of the wave and then would speed up at the top half of the wave. This issue caused me a lot of trouble until I found a solution that would allow me to achieve a perfect analog output. I had to make a separate function that only took the sine wave data values and converted them with the ratio from the ADC; this allowed for the delay to not increase significantly when calling two functions instead of one long one.



```
void tone_table(){
    int32_t temp32;
    int j;
    for(j=0;j<32;j++){
        temp32=(int32_t)sin[j]*result;
        temp32=temp32>>10;
        tone[j]=(int16_t)temp32;
        temp32=0-temp32;
        tone[j+32]=(int16_t)temp32;
    }
}
```

Figure 6 Snippet of code that took the first and second half of the sine wave input values and converted them to values adjusted by the ADC input

Aside from the many issues I faced when designing and debugging this project, I was able to implement some functions in my design that I had not before used in this course. I was able to use the Timer2 functionality of the PIC which allowed me to set every loop of gameplay to 3ms. By setting the timer to run at this speed, I could use a loop of about 10000 increments to make the length of gameplay per game session equal to approximately 30 seconds. Another thing I did to improve my design and debugging process was using the LCD and DAD to look at various outputs in my design. I used the LCD to print the output of my ADC when verifying that it worked correctly. I used the DAD to check the frequencies of my analog output sine waves both before and after adding the volume adjustment feature.

Despite everything working in the end, there are many things I think could be improved with this project. The first thing I would improve would be the timing and frequency in which LEDs are turned on during gameplay. Currently, the code is designed to randomly turn on and off LEDs in a set time range depending on the game mode, but it allows for LEDs to potentially not turn on. This issue could be fixed by adjusting the ratio of the likelihood of an LED turning on when there are currently no LEDs on. This could also be changed by allowing multiple LEDs to turn on at the same time or not allowing LEDs to turn on if they were just on. Another idea I had which I decided not to implement is adding a memory storage feature to the design; this feature would automatically save the high score of each game mode into the internal storage of the PIC and display it on the mode selection screens on the LCD. This memory storage idea is far more complex than required for this course, but I plan to add this change and the other changes I mentioned in the future to further improve this project until I get it perfect.

Appendix

```
#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <time.h>

#include <pic18.h>

#include <xc.h>

#include <pic18f47k40.h>

#define _XTAL_FREQ 64000000

#pragma config WDTE = OFF

#pragma config FEXTOSC = OFF

#pragma config RSTOSC = HFINTOSC_64MHZ

#pragma config LVP = ON


#define RS LATA4

#define EN LATA7

#define ldata LATA

#define LCD_Port TRISA


//Define Variables:

uint16_t result;

uint16_t ADC_in;

uint16_t Lower;

uint16_t Upper;
```

```
uint16_t LED0, LED1, LED2, LED3, LED4, LED5, LED6;

uint16_t SW0, SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8;

uint16_t delay_num, delay_num_min, delay_num_max, delay_num_range;

uint16_t mole_time_on;

uint16_t pick_mole, hit_mole;

int which_mole;

uint16_t LED_on_count, choose;

uint16_t mode_count;

uint16_t score, score_count, old_score;

uint16_t count, led_delay_count;

uint16_t waveselect, wavecount, select_sound;

uint16_t dacout, DAC_output;

unsigned char DAC_output_H, DAC_output_L;

int tone[64];

int arr, t;

char score_array[5];

int num_count0, num_count1, num_count2, num_count3, num_count4, num_count5,
num_count6;

int n, led_on;

int num, LCD_state, numled;

int select;

uint32_t sound_temp;


const uint16_t delay_min = 10;
const uint16_t delay_max = 150;
const uint16_t delay_range = 140;
const uint16_t mole_time_on_const = 200;
const uint16_t TC = 10000;  //***This value equal 30s (10000*3ms)
void ADC_init() {
    //use for frequency potentiometer input
    TRISBbits.TRISB3 = 1;
    ADPCH = 0x0B;
    ADCLK = 0x1F;
    ADREFbits.ADPREF = 0x0;
    ADCON0 = 0x84;

    __delay_ms(5);
}

static void ADCC_DischargeSampleCapacitor(void){
    ADPCH = 0x0C;
    __delay_ms(5);
}

void SPI_init(){
    SSP1STATbits.CKE = 0;
    SSP1CON1bits.SSPEN = 1;
    SSP1CLKPPS = 0b01001; //PPS for clock RB1
    SSP1DATPPS = 0b01000; //PPS for data RB0


    TRISBbits.TRISB0 = 0; //SDO output

    RB0PPS = 0x10;


    TRISBbits.TRISB1 = 0; //SCK output

    RB1PPS = 0x0F;


    TRISBbits.TRISB2 = 0; //~SS output


    __delay_ms(5);

}


static void CLK_Initialize(void)

{

 OSCCON1bits.NOSC = 6; /* HFINTOSC Oscillator */


 OSCFRQbits.HFFRQ = 8; /* HFFRQ 64 MHz */

}


void PORTC_init() { //LED0-LED6 and SW4

    ANSELC = 0x0;

    TRISC = 0x80; //RC7 is input, all others outputs

}
```

```
void PORTD_init() { //SW0-SW3 and SW5-SW8
   ANSELD = 0x0;
   TRISD = 0xFF; //All Port D registers are inputs (switches)
}


void timer_init() {
   T2PR = 0xF9;
   T2CONbits.ON = 0x0;
   T2CONbits.CKPS = 0x05;
   T2CONbits.OUTPS = 0x05;
   T2HLT = 0xA0;
   T2CLKCON = 0x01;
}


void LCD_Command(unsigned char cmd )
{
        ldata = (ldata & 0xF0) |(cmd>>4);  /*Send higher nibble of command first to PORT*/
        RS = 0;  /*Command Register is selected i.e.RS=0*/
        EN = 1;  /*High-to-low pulse on Enable pin to latch data*/
        NOP();
        EN = 0;
        __delay_ms(1);
   ldata = (ldata & 0xF0) | (0x0F & cmd);  /*Send lower nibble of command to PORT */
```

```
        EN = 1;

        NOP();

        EN = 0;

        __delay_ms(3);

}


void LCD_Char(unsigned char dat)

{

        ldata = (ldata & 0xF0) | (dat>>4);  /*Send higher nibble of data first to PORT*/

        RS = 1;  /*Data Register is selected*/

        EN = 1;  /*High-to-low pulse on Enable pin to latch data*/

        NOP();

        EN = 0;

        __delay_ms(3);

  ldata = (ldata & 0xF0) | (0x0F & dat);  /*Send lower nibble of data to PORT*/

        EN = 1;  /*High-to-low pulse on Enable pin to latch data*/

        NOP();

        EN = 0;

        __delay_ms(3);

}


void LCD_String(const char *msg)

{

        while((*msg)!=0)
```

```
            {

              LCD_Char(*msg);

              msg++;

    }

}


void LCD_String_xy(char row,char pos,const char *msg)

{

    char location=0;

    if(row<=1)

    {

        location=(0x80) | ((pos) & 0x0f);  /*Print message on 1st row and desired location*/

        LCD_Command(location);

    }

    else

    {

        location=(0xC0) | ((pos) & 0x0f);  /*Print message on 2nd row and desired location*/

        LCD_Command(location);

    }



    LCD_String(msg);


}
```

```
void LCD_Clear()

{

          LCD_Command(0x01);  /*clear display screen*/

   __delay_ms(3);

}


void LCD_Init()

{

   LCD_Port = 0;       /*PORT as Output Port*/

   __delay_ms(15);

   LCD_Command(0x02);  /*send for initialization of LCD

                   for nibble (4-bit) mode */

   LCD_Command(0x28);  /*use 2 line and

                   initialize 5*8 matrix in (4-bit mode)*/

          LCD_Command(0x01);  /*clear display screen*/

   LCD_Command(0x0c);  /*display on cursor off*/

          LCD_Command(0x06);  /*increment cursor (shift cursor to right)*/

}


void score_keeper(){ //need to count number of times LED is turned off by a button press

   LED_on_count--;

   score_count++;

   score = score_count;
```

```
    char array[2];     //MOVE TO LCD FUNCTION

    for(int i=0; i<2; i++){

           array[i]=0;

       }

    sprintf(array, "%.2d", score);

    LCD_Command((0xC0)|(7 & 0x0F));

    for(int j=0; j<2; j++){

        char temp = array[j];   //Temporary storage

        LCD_Char(temp); //Print digit of score value

        }

}


void turn_on_mole(){

    if(led_delay_count < delay_num){

        led_delay_count++;

    }

    else {

        led_delay_count=0;

        delay_num = rand()%(delay_num_range+1) + delay_num_min;

        pick_mole = rand()%7;


    switch(pick_mole) {

        case 0: //button 0 - LED0

            if(LED_on_count==3){
```

```
        break;  }
    else if(LED_on_count==0) {
        choose = rand()%2;
        if(choose==1){
            LED_on_count++;
            num_count0 = 0;
            PORTCbits.RC0 = 0x0;
        }
    }
    else {
        choose = rand()%2;
        if(choose==1){
            LED_on_count++;
            num_count0 = 0;
            PORTCbits.RC0 = 0x0;
        }
    }
    break;
case 1: //button 1 - LED1
    if(LED_on_count==3){
        break;  }
    else if(LED_on_count==0) {
        choose = rand()%2;
        if(choose==1){
         LED_on_count++;

         num_count1 = 0;

         PORTCbits.RC1 = 0x0;

      }

   }

   else {

      choose = rand()%2;

      if(choose==1){

         LED_on_count++;

         num_count1 = 0;

         PORTCbits.RC1 = 0x0;

      }

   }

   break;

case 2: //button 2 - LED2

   if(LED_on_count==3){

      break;  }

   else if(LED_on_count==0) {

      choose = rand()%2;

      if(choose==1){

         LED_on_count++;

         num_count2 = 0;

         PORTCbits.RC2 = 0x0;

      }
   }
   else {
      choose = rand()%2;
      if(choose==1){
         LED_on_count++;
         num_count2 = 0;
         PORTCbits.RC2 = 0x0;
      }
   }
   break;
case 3: //button 3 - LED3
   if(LED_on_count==3){
      break;  }
   else if(LED_on_count==0) {
      choose = rand()%2;
      if(choose==1){
         LED_on_count++;
         num_count3 = 0;
         PORTCbits.RC3 = 0x0;
      }
   }
   else {
      choose = rand()%2;
      if(choose==1){
         LED_on_count++;
         num_count3 = 0;
         PORTCbits.RC3 = 0x0;
      }
   }
   break;
case 4: //button 4 - LED4
   if(LED_on_count==3){
      break;  }
   else if(LED_on_count==0) {
      choose = rand()%2;
      if(choose==1){
         LED_on_count++;
         num_count4 = 0;
         PORTCbits.RC4 = 0x0;
      }
   }
   else {
      choose = rand()%2;
      if(choose==1){
         LED_on_count++;
         num_count4 = 0;
         PORTCbits.RC4 = 0x0;
      }
   }
   break;
case 5: //button 5 - LED5
   if(LED_on_count==3){
      break;  }
   else if(LED_on_count==0) {
      choose = rand()%2;
      if(choose==1){
         LED_on_count++;
         num_count5 = 0;
         PORTCbits.RC5 = 0x0;
      }
   }
   else {
      choose = rand()%2;
      if(choose==1){
         LED_on_count++;
         num_count5 = 0;
         PORTCbits.RC5 = 0x0;
      }
   }
   break;
case 6: //button 6 - LED6
   if(LED_on_count==3){
```

```
                break;  }
            else if(LED_on_count==0) {
                choose = rand()%2;
                if(choose==1){
                    LED_on_count++;
                    num_count6 = 0;
                    PORTCbits.RC6 = 0x0;
                }
            }
            else {
                choose = rand()%2;
                if(choose==1){
                    LED_on_count++;
                    num_count6 = 0;
                    PORTCbits.RC6 = 0x0;
                }
            }
            break;
    }
    }
}

int LED(numled){
    switch(numled){
```

```
case 0:
    if(PORTCbits.RC0 == 0x0){
        led_on = 1;
    }
    else{
        led_on = 0;
    }
    break;
case 1:
    if(PORTCbits.RC1 == 0x0){
        led_on = 1;
    }
    else{
        led_on = 0;
    }
    break;
case 2:
    if(PORTCbits.RC2 == 0x0){
        led_on = 1;
    }
    else{
        led_on = 0;
    }
    break;
```

```
case 3:
    if(PORTCbits.RC3 == 0x0){
        led_on = 1;
    }
    else{
        led_on = 0;
    }
    break;
case 4:
    if(PORTCbits.RC4 == 0x0){
        led_on = 1;
    }
    else{
        led_on = 0;
    }
    break;
case 5:
    if(PORTCbits.RC5 == 0x0){
        led_on = 1;
    }
    else{
        led_on = 0;
    }
    break;
```

```
      case 6:
         if(PORTCbits.RC6 == 0x0){
            led_on = 1;
         }
         else{
            led_on = 0;
         }
         break;
   }
   return led_on;
}

void button(num){
   switch (num){
      case 0:
         if(PORTDbits.RD7==0){ //button 0 is pressed
            PORTCbits.RC0 = 0x01; //turn off LED0
            score_keeper(); //increment score and decrement LED_on_count
         }
         else if(num_count0>mole_time_on){ //if button is not pressed by end time
            PORTCbits.RC0 = 0x01; //turn off LED0
            LED_on_count--; //decrement number of LEDs on
         }
         else { //button is not pressed but within time
```

```
        num_count0++;
    }
    break;
case 1:
    if(PORTDbits.RD6==0){ //button 1 is pressed
        PORTCbits.RC1 = 0x01; //turn off LED1
        score_keeper(); //increment score and decrement LED_on_count
    }
    else if(num_count1>mole_time_on){ //if button is not pressed by end time
        PORTCbits.RC1 = 0x01; //turn off LED1
        LED_on_count--; //decrement number of LEDs on
    }
    else { //button is not pressed but within time
        num_count1++;
    }
    break;
case 2:
    if(PORTDbits.RD5==0){ //button 2 is pressed
        PORTCbits.RC2 = 0x01; //turn off LED2
        score_keeper(); //increment score and decrement LED_on_count
    }
    else if(num_count2>mole_time_on){ //if button is not pressed by end time
        PORTCbits.RC2 = 0x01; //turn off LED2
        LED_on_count--; //decrement number of LEDs on
    }
    else { //button is not pressed but within time
        num_count2++;
    }
    break;
case 3:
    if(PORTDbits.RD4==0){ //button 3 is pressed
        PORTCbits.RC3 = 0x01; //turn off LED3
        score_keeper(); //increment score and decrement LED_on_count
    }
    else if(num_count3>mole_time_on){ //if button is not pressed by end time
        PORTCbits.RC3 = 0x01; //turn off LED3
        LED_on_count--; //decrement number of LEDs on
    }
    else { //button is not pressed but within time
        num_count3++;
    }
    break;
case 4:
    if(PORTDbits.RD1==0){ //button 4 is pressed ***SW4 is now SW8 -> RD1
        PORTCbits.RC4 = 0x01; //turn off LED4
        score_keeper(); //increment score and decrement LED_on_count
    }
    else if(num_count4>mole_time_on){ //if button is not pressed by end time
```

```
      PORTCbits.RC4 = 0x01; //turn off LED4
      LED_on_count--; //decrement number of LEDs on
   }
   else { //button is not pressed but within time
      num_count4++;
   }
   break;
case 5:
   if(PORTDbits.RD3==0){ //button 0 is pressed
      PORTCbits.RC5 = 0x01; //turn off LED5
      score_keeper(); //increment score and decrement LED_on_count
   }
   else if(num_count5>mole_time_on){ //if button is not pressed by end time
      PORTCbits.RC5 = 0x01; //turn off LED5
      LED_on_count--; //decrement number of LEDs on
   }
   else { //button is not pressed but within time
      num_count5++;
   }
   break;
case 6:
   if(PORTDbits.RD2==0){ //button 6 is pressed
      PORTCbits.RC6 = 0x01; //turn off LED6
      score_keeper(); //increment score and decrement LED_on_count
```

```
         }
         else if(num_count6>mole_time_on){ //if button is not pressed by end time
            PORTCbits.RC6 = 0x01; //turn off LED6
            LED_on_count--; //decrement number of LEDs on
         }
         else { //button is not pressed but within time
            num_count6++;
         }
         break;
   }
   score=score_count;
   //LCD_function(0);
}

void mole(){
   for(n=0;n<=6;n++){
      led_on = LED(n);
      if (led_on==1){
         button(n);
         //LCD_function(0);
      }
   }
}
```

```
void mode_select(){ //choose mode using SW7 (RD0)
  if(PORTDbits.RD0==0){ //if mode select button is pressed
    mode_count++;
    while(PORTDbits.RD0==1);
    if (mode_count==1 | mode_count==0){ //EASY will be default mode, mode_count starts at 0
      //EASY mode
      LCD_String_xy(1,0,"MODE: EASY  ");
      delay_num_min = delay_min;
      delay_num_max = delay_max*3;
      delay_num_range = delay_num_max - delay_num_min;
      mole_time_on = mole_time_on_const*3; //time LED stays on if not hit
      __delay_ms(250);
    }
    else if (mode_count==2){
      //MEDIUM mode
      LCD_String_xy(1,0,"MODE: MEDIUM");
      delay_num_min = delay_min;
      delay_num_max = delay_max*2;
      delay_num_range = delay_num_max - delay_num_min;
      mole_time_on = mole_time_on_const*2; //time LED stays on if not hit
      __delay_ms(250);
    }
    else if (mode_count==3){
            //HARD mode
            LCD_String_xy(1,0,"MODE: HARD  ");
            delay_num_min = delay_min;
            delay_num_max = delay_max;
            delay_num_range = delay_num_max - delay_num_min;
            mole_time_on = mole_time_on_const; //time LED stays on if not hit
            __delay_ms(250);
        }
        else {
            mode_count = 0;  //mode_count loops back around to be EASY again
        }
    }
}

void game_loop(){
    LCD_String_xy(2,0,"SCORE: ");
    for(count=0;count<=TC;count++){ //counts for the entire game play cycle
        while(PIR4bits.TMR2IF == 0);
        PIR4bits.TMR2IF = 0x0;
        //delay_num = rand()%(delay_num_range+1) + delay_num_min; //picks a delay time b/w moles turning on
        //delay_num = 20;
        //which_mole = rand()%7;
        //for(led_delay_count=0;led_delay_count<=delay_num;led_delay_count++){
         //turn_on_mole(which_mole);
      turn_on_mole();
      mole();
      //}
   }
   T2CONbits.ON = 0x0;
}

void game_end() {
   //end conditions for the game
   //resets game mode to EASY
   //toggles all game button LEDs, displays final score, plays end sound effect
   //play end sound effect
   mode_count = 0;
   //audio(2); //end sound
   LCD_String_xy(1,0,"GAME OVER!      ");
   LCD_String_xy(2,0,"FINAL SCORE: ");
   char array[2];     //MOVE TO LCD FUNCTION
   for(int i=0; i<2; i++){
        array[i]=0;
     }
   sprintf(array, "%.2d", score);
   LCD_Command((0xC0)|(13 & 0x0F));
   for(int j=0; j<2; j++){
      char temp = array[j];   //Temporary storage
      LCD_Char(temp); //Print digit of score value
      }
   audio(2); //end sound
   //LCD_function(0);
   while(PORTDbits.RD0==1);
   //PORTC ^= 0x7F; //toggles LEDs
   //__delay_ms(200);
}

void game_start(){
   //this will start the game based on which buttons are pressed (any button other than mode select)
   //mode select will be default on EASY
   //displays countdown (3...2...1...GO!) and plays starting sound effect
   PORTC = 0xFF; //turn off LEDs
   LCD_String_xy(2,0,"SELECT MODE     ");
   score_count=0;
   score=0;
   LED_on_count=0; //reset defaults for each game
   delay_num = 0;
   led_delay_count = 0;
   mode_select();
   if (PORTDbits.RD2==0 || PORTDbits.RD3==0 || PORTDbits.RD4==0 || PORTDbits.RD5==0 || PORTDbits.RD6==0 || PORTDbits.RD7==0 || PORTDbits.RD1==0){
```

```
//start conditions
//PORTC = 0xFF; //turn off LEDs
//send sound to DAC with SPI (find 3...2...1...GO! sound)
//write to LCD
LCD_Clear();
if(mode_count==1 | mode_count==0){
   LCD_String_xy(1,0,"MODE: EASY  ");
}
else if(mode_count==2){
   LCD_String_xy(1,0,"MODE: MEDIUM");
}
else if(mode_count==3){
   LCD_String_xy(1,0,"MODE: HARD  ");
}
//audio(0); //start1
LCD_String_xy(2,0,"3...");
audio(0); //start1
__delay_ms(150);
//audio(0); //start1
LCD_String_xy(2,0,"2...");
audio(0); //start1
__delay_ms(150);
//audio(0); //start1
LCD_String_xy(2,0,"1...");
```

```
        audio(0); //start1
        __delay_ms(150);
        //audio(1); //start2
        LCD_String_xy(2,0,"GO!");
        audio(1); //start1
        __delay_ms(150);
        //start game timer and begin game
        //CALL TIMER
        T2CONbits.ON = 0x01;
        game_loop();
        game_end();
    }
    else {
       // PORTC^=0x7F;
       // __delay_us(200);
        //do nothing
    }

}

uint16_t volume_ctrl(){
    ADCON0bits.GO = 1;
    __delay_ms(100);
    while(ADCON0bits.GO == 1);
    uint16_t Lower = ADRESL;

    uint16_t Upper = ADRESH;

    uint16_t value = (0x3FF&((Upper << 8)|(Lower)));

    result = value;

    return result;

}


int sin[64] = {0, 50, 100, 148, 196, 241, 284, 324,

            361, 395, 425, 451, 472, 489, 501, 509,

            511, 509, 501, 489, 472, 451, 425, 395,

            361, 324, 284, 241, 196, 148, 100, 50,

            0, -50, -100, -148, -196, -241, -284, -324,

            -361, -395, -425, -451, -472, -489, -501, -509,

            -511, -509, -501, -489, -472, -451, -425, -395,

            -361, -324, -284, -241, -196, -148, -100, -50};


void tone_table(){

    int32_t temp32;

    int j;

    for(j=0;j<32;j++){

        temp32=(int32_t)sin[j]*result;

        temp32=temp32>>10;

        tone[j]=(int16_t)temp32;

        temp32=0-temp32;
        tone[j+32]=(int16_t)temp32;
    }
}

uint16_t sound_out(arr){
    //result=volume_ctrl();

    //sound_temp=(uint32_t)sin[arr]*result;
    //sound_temp=sound_temp>>10;
    dacout=tone[arr]+512;
    //dacout = dacout+512;
    //dacout = sinnew[arr];
    //dacout = (dacout > 1023 ? 1023: dacout);

    dacout = dacout<<2 & 0x0FFC;
    dacout = dacout | 0x9000U;          // configure for DAC, write data to DAC A and update

    return dacout;
    //return result;
}

audio(select) {
    result = volume_ctrl();
    tone_table();
```

```
switch(select){
    case 0: //start1
        for(t=0;t<=600;t++){ //length of sound=300ms
            for(wavecount = 0; wavecount<64; wavecount++) {
            DAC_output = (unsigned int)sound_out(wavecount);
            //DAC_output = (unsigned int)DAC_output;

            DAC_output_L = DAC_output;
            DAC_output = DAC_output >> 8;
            DAC_output_H = DAC_output;

            SSP1CON1 = 0x21;
            SSP1STAT = 0x40; //configures clock

            LATBbits.LATB2 = 0;

            SSP1BUF = DAC_output_H;
            while(!SSP1STATbits.BF);
            PIR3bits.SSP1IF = 0;
            SSP1BUF = DAC_output_L;
            while(!SSP1STATbits.BF);
            LATBbits.LATB2 = 1;
            SSP1CON1 = 0x00;
        __delay_us(10); //sets frequency to 1000Hz

        }

    }

    break;

case 1: //start2

    for(t=0;t<=1000;t++){ //length of sound=300ms

        for(wavecount = 0; wavecount<64; wavecount++) {

        DAC_output = sound_out(wavecount);

        DAC_output = (unsigned int)DAC_output;

        //DAC_output = 0x9A55; //test value


        DAC_output_L = DAC_output;

        DAC_output = DAC_output >> 8;

        DAC_output_H = DAC_output;


        SSP1CON1 = 0x21;

        SSP1STAT = 0x40; //configures clock


        LATBbits.LATB2 = 0;


        SSP1BUF = DAC_output_H;

        while(!SSP1STATbits.BF);

        PIR3bits.SSP1IF = 0;

        SSP1BUF = DAC_output_L;
        while(!SSP1STATbits.BF);
        LATBbits.LATB2 = 1;
        SSP1CON1 = 0x00;

        __delay_us(5); //sets frequency to 1500Hz
        }
      }
      break;
    case 2: //end
      for(t=0;t<=1700;t++){ //length of sound=1.5s
        for(wavecount = 0; wavecount<64; wavecount++) {
        DAC_output = sound_out(wavecount);
        DAC_output = (unsigned int)DAC_output;
        //DAC_output = 0x9A55; //test value

        DAC_output_L = DAC_output;
        DAC_output = DAC_output >> 8;
        DAC_output_H = DAC_output;

        SSP1CON1 = 0x21;
        SSP1STAT = 0x40; //configures clock

        LATBbits.LATB2 = 0;
        SSP1BUF = DAC_output_H;
        while(!SSP1STATbits.BF);
        PIR3bits.SSP1IF = 0;
        SSP1BUF = DAC_output_L;
        while(!SSP1STATbits.BF);
        LATBbits.LATB2 = 1;
        SSP1CON1 = 0x00;

        __delay_us(7); //set frequency to 1250Hz
        }
      }
      break;
  }
}

int main(void) {
  ADC_init();
  SPI_init();
  CLK_Initialize();
  PORTC_init();
  PORTD_init();
  timer_init();
  LCD_Init();
```

```
ADCC_DischargeSampleCapacitor();

ADPCH = 0x0B;


srand(time(NULL));


while(1){

    //T2CON = 0xA3;

    //PORTC ^= 0x7F;

    //__delay_us(200);

    game_start();

    //audio(0);

    //audio(1);

    //audio(2);

    //result=volume_ctrl();

    /*char array[4]; //Storage array

   //Clear array

   for(int i=0; i<4; i++){

       array[i]=0;

   }

    sprintf(array, "%.4d", result);

    LCD_Command((0xC0)|(7 & 0x0F));

    for(int j=0; j<4; j++){

            char temp = array[j];   //Temporary storage

            LCD_Char(temp); //Print digit of score value
```

```
      }*/

   }



}
```

References

Interfacing LCD 16x2 in 4-bit mode with PIC18F4550. ElectronicWings. (n.d.). Retrieved September 6, 2022, from https://www.electronicwings.com/pic/interfacing-lcd-16x2-in-4-bit-mode-with-pic18f4550-

Microchip Technology Inc. (2020). MPLAB Snap User's Guide.

Microchip Technology Inc. (2018). PIC18(L)F27/47K40 Data Sheet.

Linear Technology Corporation. (n.d.). LTC1661 – Micropower Dual 10-Bit DAC in MSOP. Milpitas.

Texas Instruments. (1999, December). TLV4110, TLV4111, TLV4112, TLV4113 Family of High Output Drive Operational Amplifiers with Shutdown Datasheet. Dallas.