Final Project: Whack-A-Mole

Owen Helfrich, Nita Oktavia, Xuewei Bai

Table of Contents

Introduction:

In this project, we will be designing a Whack-a-Mole game that uses PWMs and PRS to control the grid of LED. When an LED lights up, the user presses the keypad button that corresponds to that LED. When the user clicks the right button, the speaker will make one tone; when the user presses the wrong button, a different tone will be made. The user's score as well as the highest score from the history recorded will be displayed on the LCD when the game is over. After the game begins, users can play it 20 more times by pressing the button in accordance with the LED light. However, after pressing the button 20 times, the game is over. Users just need to restart the game in order to play another game.

Hardware Description:

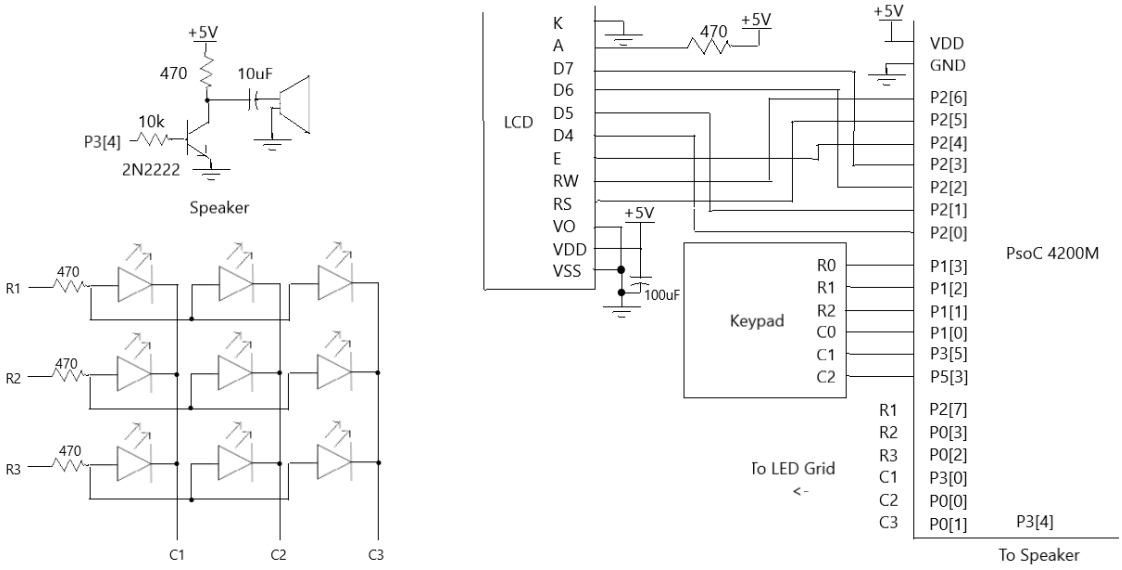

Figure 1: Hardware Diagram

Hardware Components:

- PSoC 4200M (CY8KIT-044) - $25
- 4x4 Keypad - $1.95
- 9 LEDs - $2.50 for 10
- 2x16 LCD - $34.95
- Speaker - $1.49
- 6 470Ω Resistors - $4.29 for 10
- Breadboard - $7.95
- 22-Gauge Single-Strand Coated Copper Wire - $8.85 for 25 ft.

Configuration:

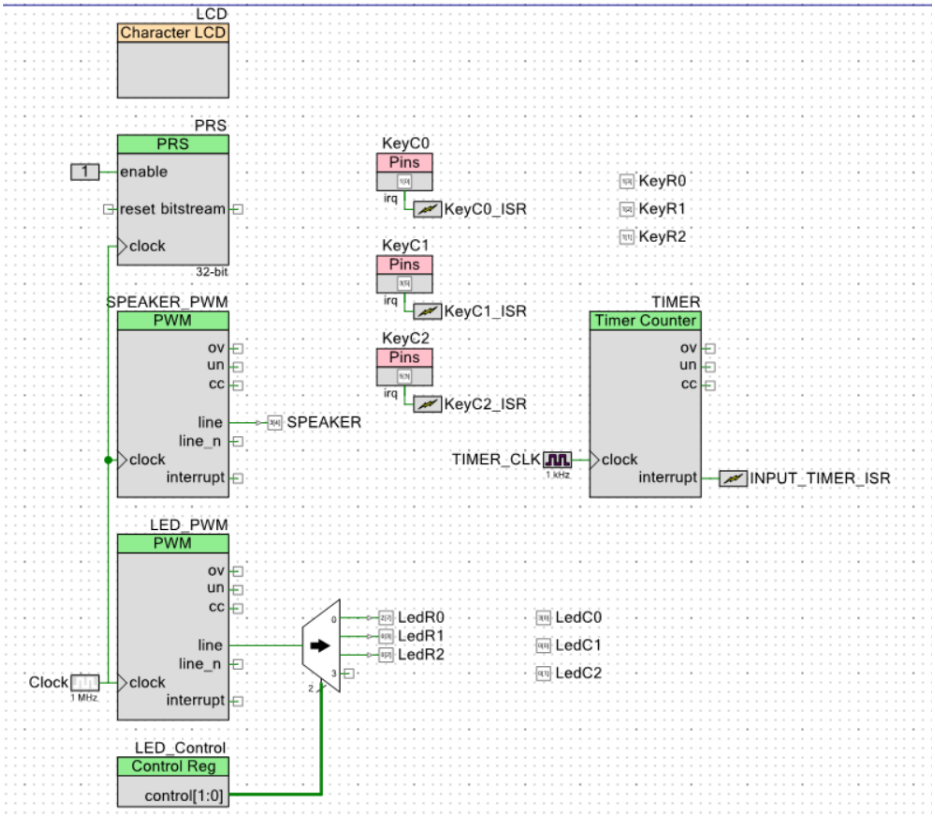

Figure 2: Top Design

- Follow the Top Design for the module that we will use

- Set the Pins and Ports according to the figure below

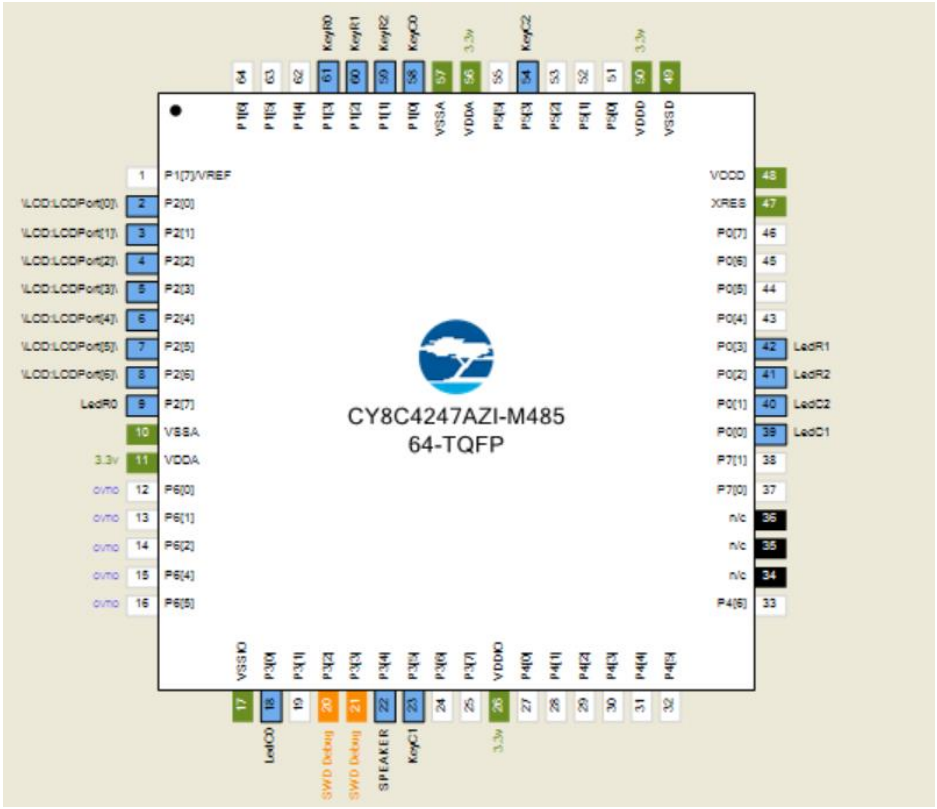

Figure 3: Pins

	Name	Port	Pin
	\LCD:LCDPort[6:0]\	P2[6:0]	8…2
	KeyC0	P1[0]	58
	KeyC1	P3[5]	23
	KeyC2	P5[3]	54
	KeyR0	P1[3]	61
	KeyR1	P1[2]	60
	KeyR2	P1[1]	59
	LedC0	P3[0]	18
	LedC1	P0[0]	39
	LedC2	P0[1]	40
	LedR0	P2[7]	9
	LedR1	P0[3]	42
	LedR2	P0[2]	41
	SPEAKER	P3[4]	22

Figure 4: Ports

- Configure the LCD Custom Character Set to None
- Rename the speaker module to SPEAKER and set it to digital output and check the HW Connection button. Set the drive mode to Strong Drive
- Configure the LED Demultiplexer and set the NumOutputTerminals to 4 and TerminalWidth to 1. Rename it to demux_1
- Set the LED Row Pin configuration to digital output and check the HW Connection box and set the drive mode to Strong Drive
- Set the LED Row Pin configuration to digital output and check the HW Connection box and set the drive mode to Strong Drive
- Configure the LED_Control Register outputs into 2 and set both of it to Direct mode
- In general box of the Keypad Column Pin configuration, set it to digital input and set the drive mode to Strong Drive. In input box, change the interrupt to Rising Edge and check the Dedicated Interrupt
- Rename the Keypad Column Pin Interrupt to KeyC0_ISR and set the InterruptType to DERIVED
- Configure the Keypad Row Pin to digital output and set the drive mode to Strong Drive
- In the Timer configuration, choose timer/counter
- Set the Timer Clock to 1 kHz and align it to 48 MHz
- Rename the Timer Interrupt to INPUT_TIMER_ISR and set the InterruptType to DERIVED
- Your module configurations should look like these

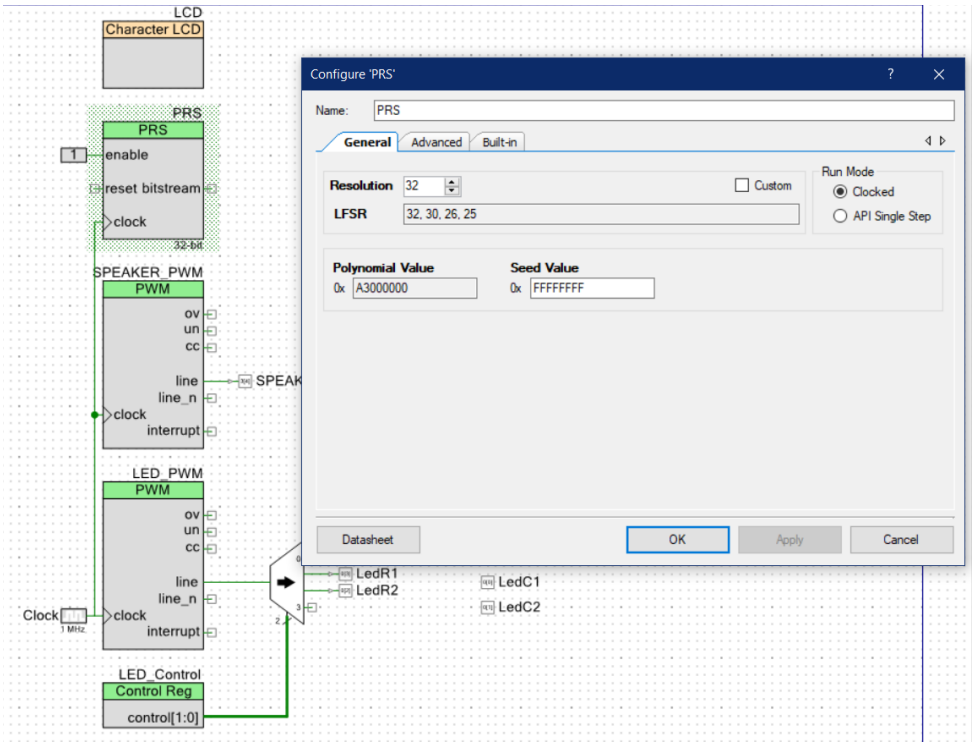

Figure 5: PRS Configuration

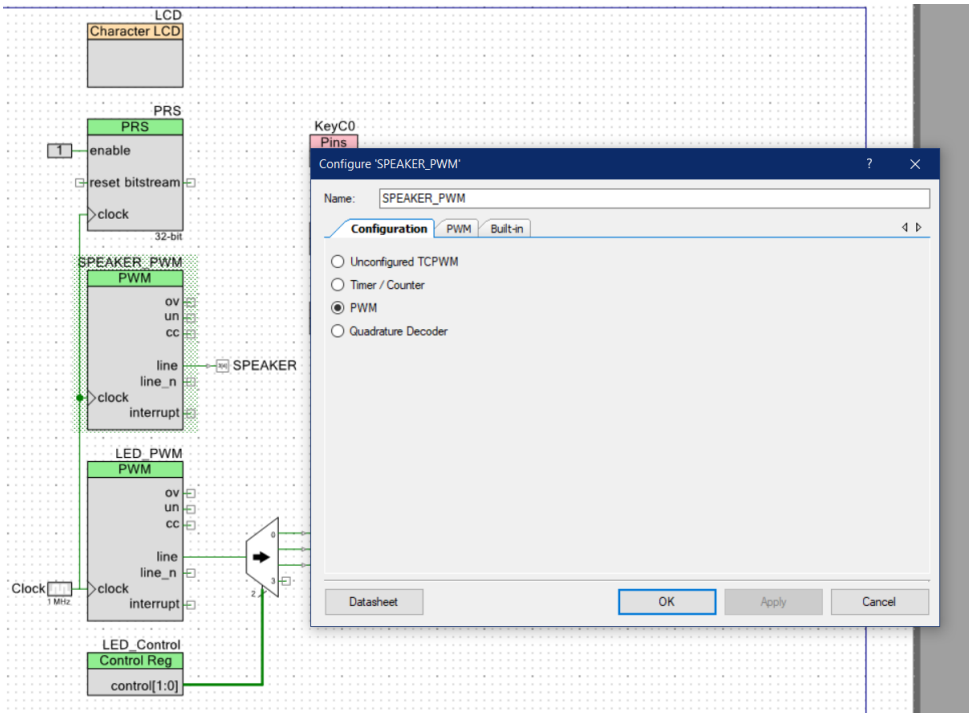

Figure 6: Speaker PWM Configuration

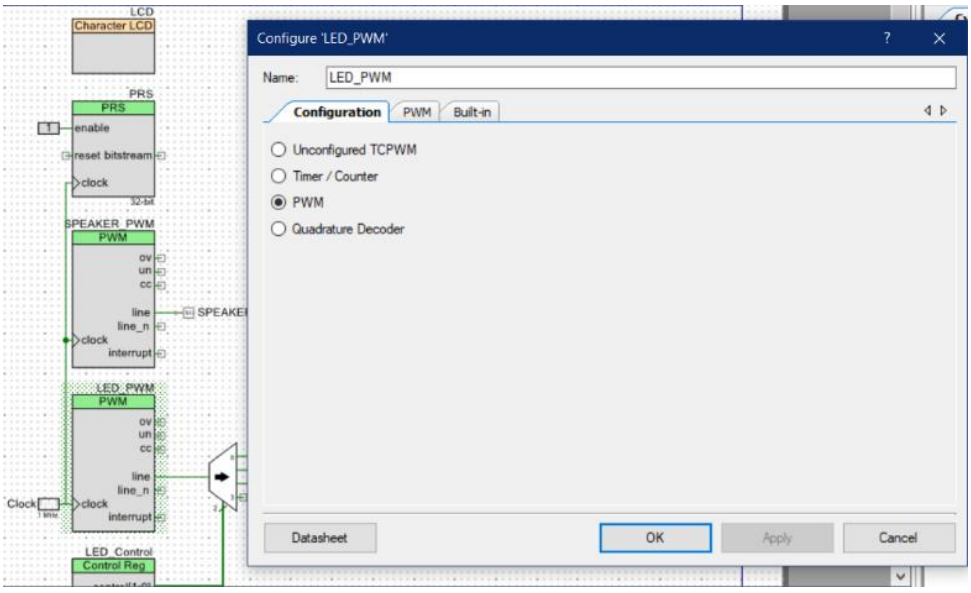

Figure 7: LED PWM Configuration

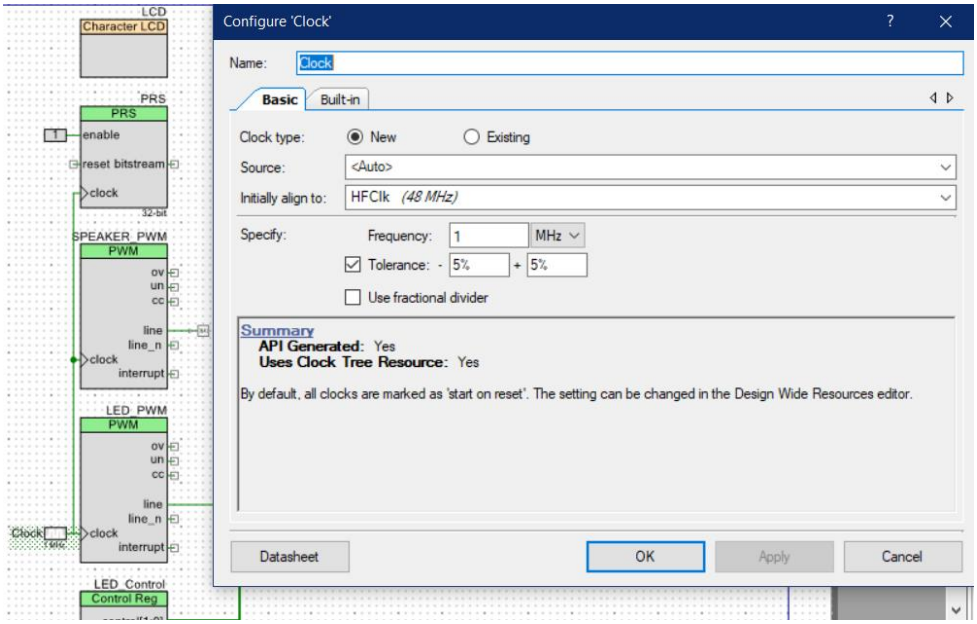

Figure 8: PWM and PRS Clock Configuration

Instance Name	Interrupt Number	Priority (0 - 3)
INPUT_TIMER_ISR	20	0
KeyC0_ISR	1	3
KeyC1_ISR	3	3
KeyC2_ISR	0	3

Figure 9: Input Priority Setting Configuration

Module Lists:

- Character LCD
- 3 Digital Input Pins (for keypad)
- 9 Digital Output Pins (6 for LEDs, 3 for keypad)
- PRS
- PWM (Timer, LEDs, Speaker)
- 2 Clock
- 4 Interrupts (for each keypad row, one for the timer)

Codes:

First, the user presses a button to start the game. The PRS generates 60 random numbers between 0 and 99 which are stored into an array.

```
start_game();       // Executes the start_game() function, plays the starting
tones

        for(int i = 0; i < 60 ; i++)            // Reads the PRS 60 times
        {
            mole_info[i] = PRS_Read() % 100;    // Input a value, range 0-99,
into the mole_info[] array
            CyDelay(10);                        // Delay to avoid inputting
multiples of the same value
        }
```

Then, the array is read three times – one number is modulated by 9 and used to determine which LED turns on, the next is used to determine the brightness of the LED via the PWM, and the last is multiplied by 50 and used to determine how long the LED is on for, in milliseconds.

```
while(rounds < 20)      // For 20 rounds, do the following:
        {
            LED = mole_info[1+(3*mole_ctr)] % 9;                // LED number
is the 1st, 4th, 7th ... number in array
                                                                //      Max
is 8, min is 0
            brightness = (mole_info[2+(3*mole_ctr)] % 90) + 10; // Brightness
is the 2nd, 5th, 8th ... number in the array
                                                                //      Max
is 99, min is 10
            duration = (mole_info[3+(3*mole_ctr)]+20) * 25;     // Duration
is the 3rd, 6th, 9th ... number in the array
                                                                //  Max is
2.975s, min is 500ms
            led_sel(LED, brightness, duration);                 // Executes
the led_sel() function, turns on the appropriate LED

            mole_ctr++;         // Increase the number of moles seen by 1
            rounds++;           // Increase the number of rounds finished by
1
        }
```

The PSoC converts the number corresponding to the LED into the appropriate rows and columns, writing the row pin to 5V and the column pin to ground, while the other row pins are set to ground and the other column pins are set to 5V. This way, the intended LED will be the only one to have 5V on the anode and 0V on the cathode; current will flow through this LED and not the others.

```
switch(led)                     // Determine which LED row will be high and
column low - turns on LED
    {
        case 0:     // Top row, leftmost column LED
            LED_Control_Write(2);   // Writes the leftmost column high (board
was wired at 90 deg and it's too much hassle to change)
```

```
            SPEAKER_PWM_WritePeriod((int)((1/led0)/(0.000001)));        //
Plays a note when the LED turns on, specific to this LED
            SPEAKER_PWM_WriteCompare((int)(((1/led0)/(0.000001))/2));

            CyDelay(50);

            LedC0_Write(0);         // Writes the top row low to turn on the
desired LED
            LedC1_Write(1);         // Writes the middle row high to prevent
these LEDs from turning on
            LedC2_Write(1);         // Writes the bottom row to high to
prevent these LEDs from turning on

            break;
        case 1:     // Top row, middle column LED
            LED_Control_Write(1);

            SPEAKER_PWM_WritePeriod((int)((1/led1)/(0.000001)));
            SPEAKER_PWM_WriteCompare((int)(((1/led1)/(0.000001))/2));

            CyDelay(50);

            LedC0_Write(0);
            LedC1_Write(1);
            LedC2_Write(1);
            break;
        case 2:     // Top row, rightmost column LED
            LED_Control_Write(0);

            SPEAKER_PWM_WritePeriod((int)((1/led2)/(0.000001)));
            SPEAKER_PWM_WriteCompare((int)(((1/led2)/(0.000001))/2));

            CyDelay(50);

            LedC0_Write(0);
            LedC1_Write(1);
            LedC2_Write(1);
            break;
        case 3:     // Middle row, leftmost column LED
            LED_Control_Write(2);

            SPEAKER_PWM_WritePeriod((int)((1/led3)/(0.000001)));
            SPEAKER_PWM_WriteCompare((int)(((1/led3)/(0.000001))/2));

            CyDelay(50);

            LedC0_Write(1);
            LedC1_Write(0);
            LedC2_Write(1);
            break;
        case 4:     // Middle row, middle column LED
            LED_Control_Write(1);

            SPEAKER_PWM_WritePeriod((int)((1/led4)/(0.000001)));
            SPEAKER_PWM_WriteCompare((int)(((1/led4)/(0.000001))/2));

            CyDelay(50);
```

```
        LedC0_Write(1);
        LedC1_Write(0);
        LedC2_Write(1);
        break;
    case 5:     // Middle row, rightmost column LED
        LED_Control_Write(0);

        SPEAKER_PWM_WritePeriod((int)((1/led5)/(0.000001)));
        SPEAKER_PWM_WriteCompare((int)(((1/led5)/(0.000001))/2));

        CyDelay(50);

        LedC0_Write(1);
        LedC1_Write(0);
        LedC2_Write(1);
        break;
    case 6:     // Bottom row, leftmost column LED
        LED_Control_Write(2);

        SPEAKER_PWM_WritePeriod((int)((1/led6)/(0.000001)));
        SPEAKER_PWM_WriteCompare((int)(((1/led6)/(0.000001))/2));

        CyDelay(50);

        LedC0_Write(1);
        LedC1_Write(1);
        LedC2_Write(0);
        break;
    case 7:     // Bottom row, middle column LED
        LED_Control_Write(1);

        SPEAKER_PWM_WritePeriod((int)((1/led7)/(0.000001)));
        SPEAKER_PWM_WriteCompare((int)(((1/led7)/(0.000001))/2));

        CyDelay(50);

        LedC0_Write(1);
        LedC1_Write(1);
        LedC2_Write(0);
        break;
    case 8:     // Bottom row, rightmost column LED
        LED_Control_Write(0);

        SPEAKER_PWM_WritePeriod((int)((1/led8)/(0.000001)));
        SPEAKER_PWM_WriteCompare((int)(((1/led8)/(0.000001))/2));

        CyDelay(50);

        LedC0_Write(1);
        LedC1_Write(1);
        LedC2_Write(0);
        break;
}
```

While an LED is on, the PSoC writes the corresponding keypad row pin to VDD and waits for an interrupt from the button in the correct column. If the user presses the correct button in the given timeframe, a score counter is incremented and the speaker plays a success tone. If the user presses any other button, the speaker plays a failure tone. This process repeats 20 times.

```
switch(led)     // Checks if the correct button is pressed for a given mole
        {
            case 0:     // For the top row, left-most column LED
                if((KeyR0_Read() == 1) && (C0_flag == 1))   // If the top row
is strobed and the correct button is pressed
                {
                    SPEAKER_PWM_WritePeriod((int)hit);    // Set the speaker
period to the hit tone
                    SPEAKER_PWM_WriteCompare((int)hit/2); // Set the compare
counter to half the period

                    C0_flag = 0;                          // Reset the column
flag
                    score = score + 400;                  // Increase the
score
                    whacked = 1;                          // Notes the mole
as whacked, will not display it anymore

                    CyDelay(50);                          // Delay so the
speaker can actually play

                    SPEAKER_PWM_WritePeriod(0);           // Turn off the
speaker
                    SPEAKER_PWM_WriteCompare(0);
                }
                else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
// If the incorrect button is pressed
                {
                    SPEAKER_PWM_WritePeriod((int)miss);    // Set the speaker
period to the miss tone
                    SPEAKER_PWM_WriteCompare((int)miss/2); // Set the compare
counter to half the period

                    CyDelay(50);                           // Delay so the
speaker can play

                    C0_flag = 0;                           // Reset the
column flags
                    C1_flag = 0;
                    C2_flag = 0;

                    SPEAKER_PWM_WritePeriod(0);            // Turn off the
speaker
                    SPEAKER_PWM_WriteCompare(0);
                }
                break;
            case 1:     // For the top row, middle column LED
                if((KeyR0_Read() == 1) && (C1_flag == 1))
                {
                    SPEAKER_PWM_WritePeriod((int)hit);
                    SPEAKER_PWM_WriteCompare((int)hit/2);
```

```
            C1_flag = 0;
            score = score + 400;
            whacked = 1;

            CyDelay(50);

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)miss);
            SPEAKER_PWM_WriteCompare((int)miss/2);

            CyDelay(50);

            C0_flag = 0;
            C1_flag = 0;
            C2_flag = 0;

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        break;
    case 2:     // For the top row, rightmost column LED
        if((KeyR0_Read() == 1) && (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)hit);
            SPEAKER_PWM_WriteCompare((int)hit/2);

            C2_flag = 0;
            score = score + 400;
            whacked = 1;

            CyDelay(50);

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)miss);
            SPEAKER_PWM_WriteCompare((int)miss/2);

            CyDelay(50);

            C0_flag = 0;
            C1_flag = 0;
            C2_flag = 0;

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        break;
    case 3:     // For the middle row, leftmost column LED
        if((KeyR1_Read() == 1) && (C0_flag == 1))
```

```
        {
            SPEAKER_PWM_WritePeriod((int)hit);
            SPEAKER_PWM_WriteCompare((int)hit/2);

            C0_flag = 0;
            score = score + 400;
            whacked = 1;

            CyDelay(50);

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)miss);
            SPEAKER_PWM_WriteCompare((int)miss/2);

            CyDelay(50);

            C0_flag = 0;
            C1_flag = 0;
            C2_flag = 0;

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        break;
    case 4:     // For the middle row, middle column LED
        if((KeyR1_Read() == 1) && (C1_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)hit);
            SPEAKER_PWM_WriteCompare((int)hit/2);

            C1_flag = 0;
            score = score + 400;
            whacked = 1;

            CyDelay(50);

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)miss);
            SPEAKER_PWM_WriteCompare((int)miss/2);

            CyDelay(50);

            C0_flag = 0;
            C1_flag = 0;
            C2_flag = 0;

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
```

```
    break;
case 5:     // For the middle row, rightmost column LED
    if((KeyR1_Read() == 1) && (C2_flag == 1))
    {
        SPEAKER_PWM_WritePeriod((int)hit);
        SPEAKER_PWM_WriteCompare((int)hit/2);

        C2_flag = 0;
        score = score + 400;
        whacked = 1;

        CyDelay(50);

        SPEAKER_PWM_WritePeriod(0);
        SPEAKER_PWM_WriteCompare(0);
    }
    else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
    {
        SPEAKER_PWM_WritePeriod((int)miss);
        SPEAKER_PWM_WriteCompare((int)miss/2);

        CyDelay(50);

        C0_flag = 0;
        C1_flag = 0;
        C2_flag = 0;

        SPEAKER_PWM_WritePeriod(0);
        SPEAKER_PWM_WriteCompare(0);
    }
    break;
case 6:     // For the bottom row, leftmost column LED
    if((KeyR2_Read() == 1) && (C0_flag == 1))
    {
        SPEAKER_PWM_WritePeriod((int)hit);
        SPEAKER_PWM_WriteCompare((int)hit/2);

        C0_flag = 0;
        score = score + 400;
        whacked = 1;

        CyDelay(50);

        SPEAKER_PWM_WritePeriod(0);
        SPEAKER_PWM_WriteCompare(0);
    }
    else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
    {
        SPEAKER_PWM_WritePeriod((int)miss);
        SPEAKER_PWM_WriteCompare((int)miss/2);

        CyDelay(50);

        C0_flag = 0;
        C1_flag = 0;
        C2_flag = 0;
```

```
            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        break;
    case 7:     // For the bottom row, middle column LED
        if((KeyR2_Read() == 1) && (C1_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)hit);
            SPEAKER_PWM_WriteCompare((int)hit/2);

            C1_flag = 0;
            score = score + 400;
            whacked = 1;

            CyDelay(50);

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)miss);
            SPEAKER_PWM_WriteCompare((int)miss/2);

            CyDelay(50);

            C0_flag = 0;
            C1_flag = 0;
            C2_flag = 0;

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        break;
    case 8:     // For the bottom row, rightmost column LED
        if((KeyR2_Read() == 1) && (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)hit);
            SPEAKER_PWM_WriteCompare((int)hit/2);

            C2_flag = 0;
            score = score + 400;
            whacked = 1;

            CyDelay(50);

            SPEAKER_PWM_WritePeriod(0);
            SPEAKER_PWM_WriteCompare(0);
        }
        else if((C0_flag == 1) || (C1_flag == 1) || (C2_flag == 1))
        {
            SPEAKER_PWM_WritePeriod((int)miss);
            SPEAKER_PWM_WriteCompare((int)miss/2);

            CyDelay(50);

            C0_flag = 0;
```

```
                    C1_flag = 0;
                    C2_flag = 0;

                    SPEAKER_PWM_WritePeriod(0);
                    SPEAKER_PWM_WriteCompare(0);
                }
                break;
        }
        i++;                // Increase the row strobing control variable
```

Finally, the user's score and the high score are displayed on the LCD screen. The user is given the opportunity to play again by pressing a button.

```
if(score > high_score)  // Updates the high score
        {
            high_score = score;
        }

        LCD_Position(0,0);
        LCD_PrintString("Score: ");
        LCD_PrintNumber(score);             // Prints the score for the
session

        LCD_Position(1,0);
        LCD_PrintString("High Score: ");
        LCD_PrintNumber(high_score);        // Prints the high score

        mole_ctr = 0;       // Reset the number of moles seen
        rounds = 0;         // Reset the number of rounds finished
        score = 0;          // Reset the session score

        C0_flag = 0;        // Reset the key column flags
        C1_flag = 0;
        C2_flag = 0;
```

Writing Interrupt Signal to HIGH function
(The codes in KeyC0_ISR and KeyC1_ISR and KeyC2_ISR are all the same)

```
/***

*  Place your includes, defines and code here

**/

/* #START KeyC0_ISR_intc /

#include "KeyC0.h"

volatile int C0_flag = 0;

/ #END /CY_ISR(KeyC0_ISR_Interrupt)

{

    #ifdef KeyC0_ISR_INTERRUPT_INTERRUPT_CALLBACK
```

```
        KeyC0_ISR_Interrupt_InterruptCallback();
    #endif / KeyC0_ISR_INTERRUPT_INTERRUPT_CALLBACK /

    /  Place your Interrupt code here. /
    / #START KeyC0_ISR_Interrupt /
    CyDelay(200);
    if(KeyC0_Read() == 1)
    {
        C0_flag = 1;
    }

    KeyC0_ClearInterrupt();
    / #END */
}
```

Writing the Timer function

```
/***
*  Place your includes, defines and code here
**/
/* #START INPUT_TIMER_ISR_intc /
    #include "TIMER.h"
    extern volatile int timeout;
/ #END */
CY_ISR(INPUT_TIMER_ISR_Interrupt)
{
    #ifdef INPUT_TIMER_ISR_INTERRUPT_INTERRUPT_CALLBACK
        INPUT_TIMER_ISR_Interrupt_InterruptCallback();
    #endif /* INPUT_TIMER_ISR_INTERRUPT_INTERRUPT_CALLBACK /

    /  Place your Interrupt code here. /
    / #START INPUT_TIMER_ISR_Interrupt /
    timeout = 1;
```

```
    TIMER_ClearInterrupt(TIMER_INTR_MASK_TC);
    / #END */
}
```

Results with Pictures:

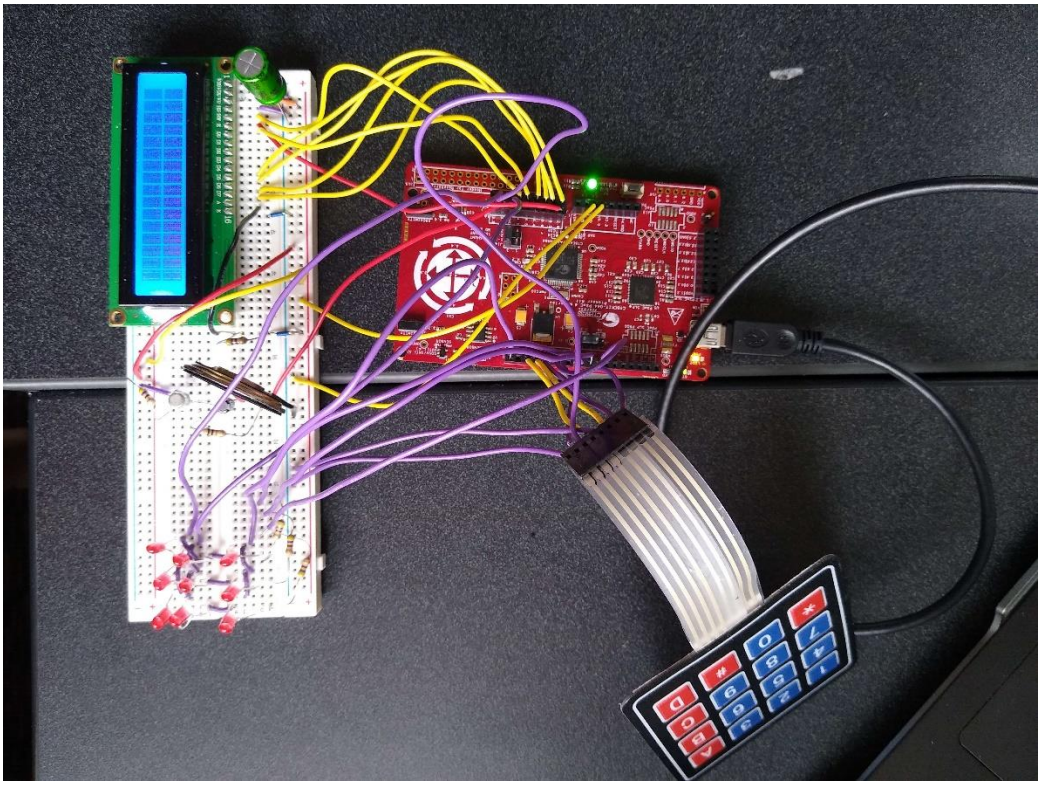

Figure 10: Before the Game Starts

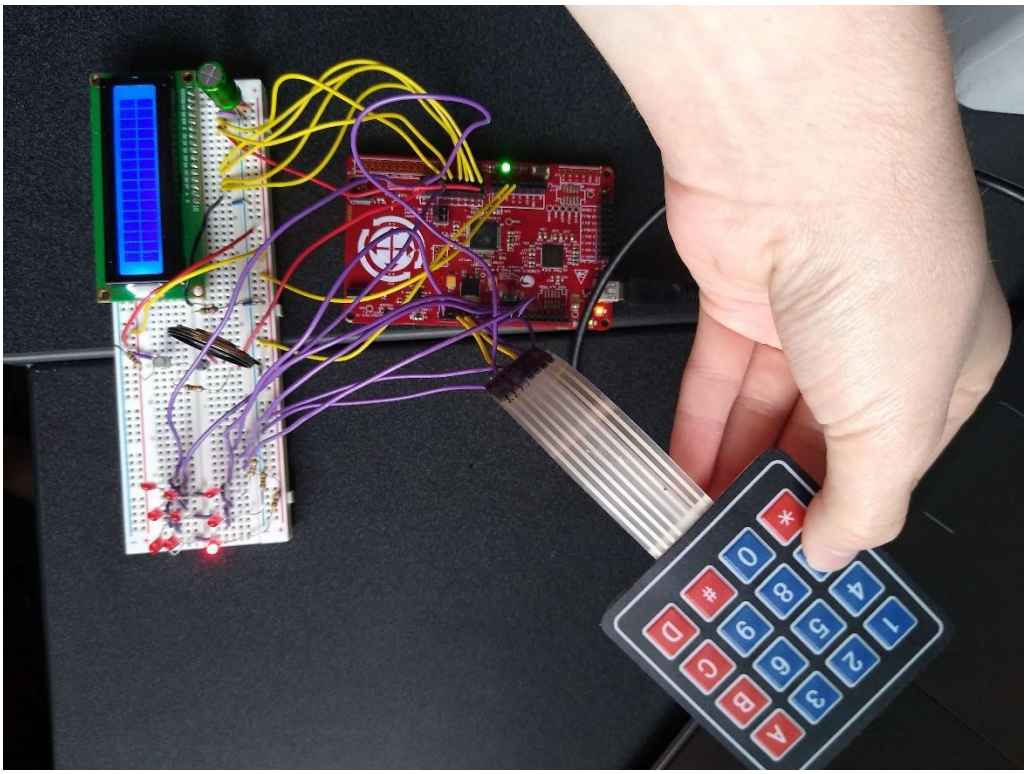

Figure 11: During the Game

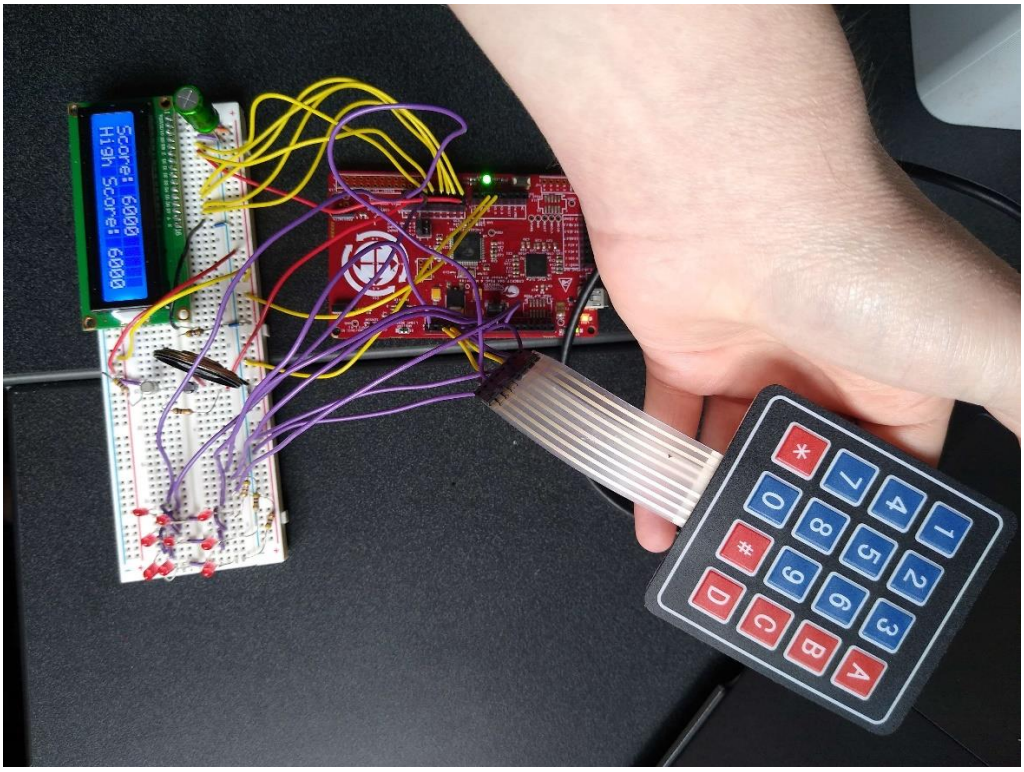

Figure 12: Game Over

Conclusion:

Overall, the lab was successful and the codes and hardware were working. If we could change anything from this project, we would choose a different kind of keypad, because the one that we have right now is not working properly as it is a bit flimsy. Instead, we would like to use 9 buttons. We would also like to put more delay from the previous game to the next one, so that we can have more time to see the score that is shown on the LCD. Delay between the noise when we press the right button and the wrong one would also be a nice improvement to our codes.

Table of Figures: