Final Project

Star Wars Slot Machine

Lab Completed By: Alex Kuzner & Chase Mulder

Lab Section 901

Professor: Trevor Ekins

Date Submitted: 8/2/19

Table Of Contents

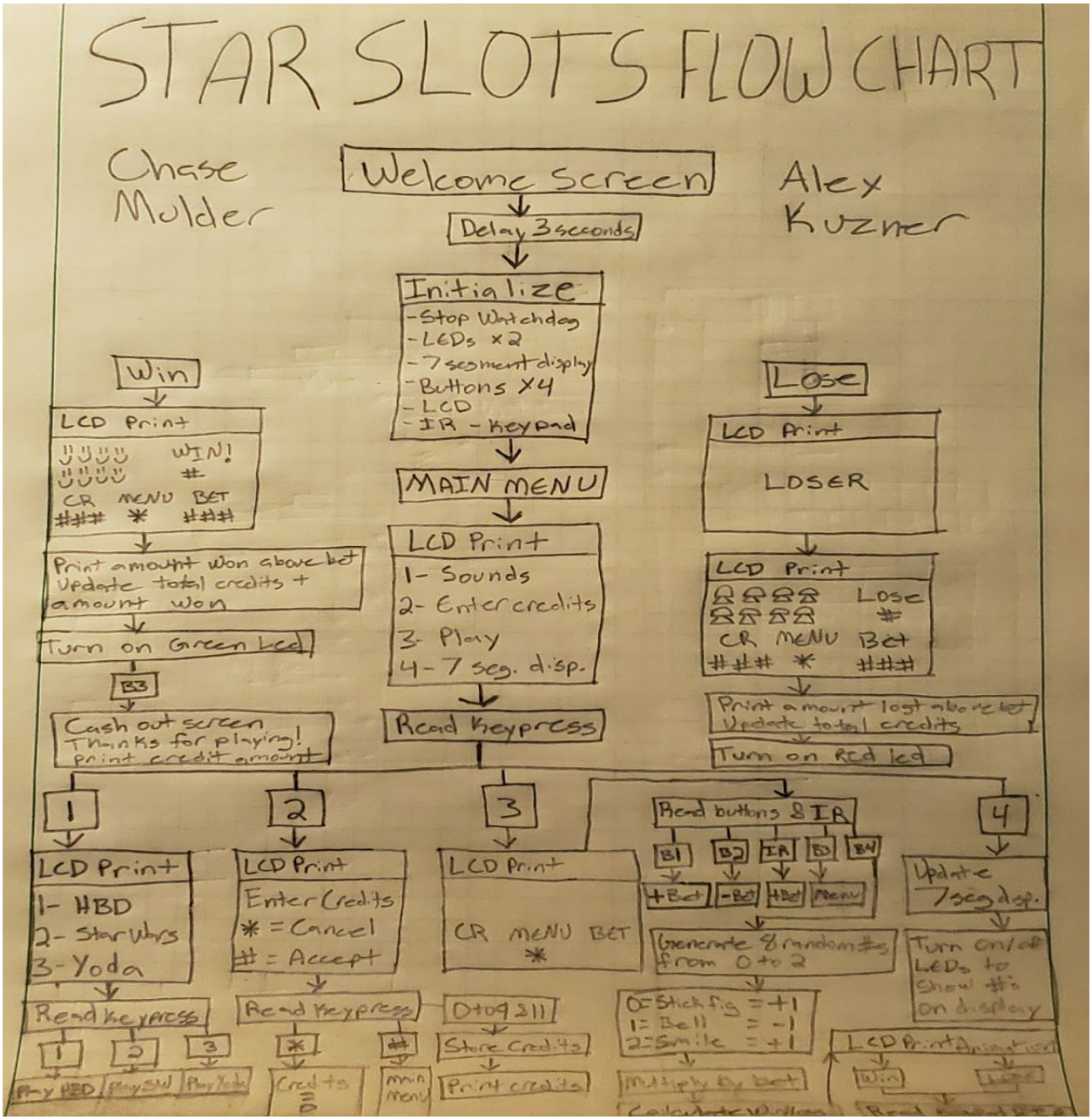
Objective:

The final project revolves around building and programming a slot machine game using the many skills learned throughout the semester. In order to complete the final project many requirements are set. The slot machine game must be programmed and used with many different pieces of hardware including: LCD, keypad, pushbuttons, RGB LED/LED's, speaker, PWM, potentiometer, 7- Segment display and an infrared sensor. Each piece has a role to play in the functionality of the game. The objective is to put the knowledge and skills learned this semester to build a whole functioning system. In doing that the game should be programmed to be able to play through like a normal slot machine. In the beginning, the LCD should display the name of the game and the names of the students. Then the next step is the display of the menu on the LCD, this should include a list of sections including adding credits, playing the game, sounds, and the seven segment contrast control. The game requires a form of winning and losing with the basis of shuffled customized characters on the LCD screen. When playing you cannot bet more than five and cannot win with a zero credit bet and whether you win or lose a sound and a green or red light should activate. Also when required a cash out screen is available when done playing the game. In the credit menu the maximum credits should be 999. In sounds there should be a minimum of two unique sounds. The seven segment display depends on the potentiometer and displays the contrast whether its high or low. In order to get a 100% two additional features must be added and two used frequently is a nice slot machine box and an IR sensor that reads when a coin is

deposited. Then the 100% can be acquired with all the necessary requirements and the additional two features.

Procedure:

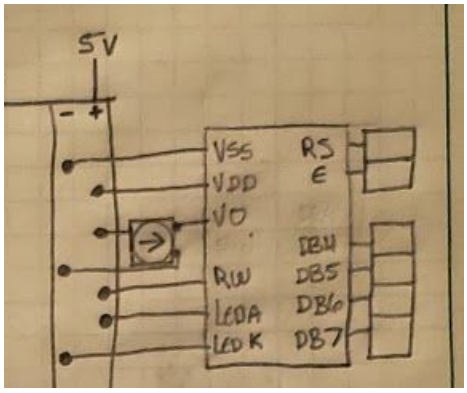
Making a slot machine requires many steps involving the programming and interfacing. To logically step by step think out what had to be done to create a fully functioning slot machine game a flow chart was fabricated:

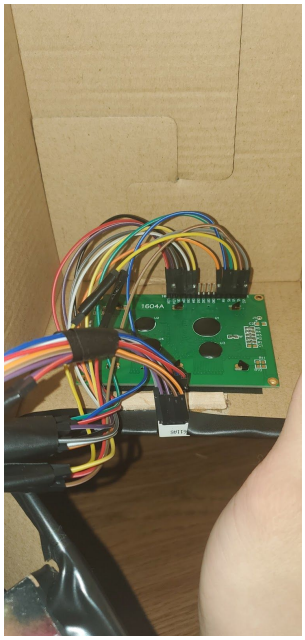


The search for a proper unit to acustom the slot machine started at home depot. After walking through every isle of homedepot multiple times the perfect size box was found to fit and display all of the hardware of the slot machine. The box was also wrapped in our theme, Star Wars which conveys the slot machines fun game like look.

To start, the first requirement is to display the name of the final project and the names of the students on the LCD. Then after the title display the slot machine revolves around the LCD screen and its display. The game cannot be played without it and most commands involve it. The LCD has to be wired to the MSP432 to be able to be written to in all parts of the game. It is used to display things such as the menus, game interface and much more.

Here is the circuit for the LCD screen:





Here is how the GPIO pins (left blank in circuit diagram) are initialized to port # and bit # on the MSP432:

```
//LCD
uint8_t LEDrs = BIT0;
P4SEL1 &= ~LEDrs;
P4SEL0 &= ~LEDrs; //RS
P4DIR |= LEDrs;
P4OUT &= ~LEDrs;
uint8_t LEDe = BIT1;
P4SEL1 &= ~LEDe;
P4SEL0 &= ~LEDe; //E
P4DIR |= LEDe;
P4OUT &= ~LEDe;
uint8_t LEDdb4 = BIT4;
P4SEL1 &= ~LEDdb4;
P4SEL0 &= ~LEDdb4; //DB4
P4DIR |= LEDdb4;
P4OUT &= ~LEDdb4;
uint8_t LEDdb5 = BIT5;
P4SEL1 &= ~LEDdb5;
P4SEL0 &= ~LEDdb5; //DB5
P4DIR |= LEDdb5;
P4OUT &= ~LEDdb5;
uint8_t LEDdb6 = BIT6;
P4SEL1 &= ~LEDdb6;
P4SEL0 &= ~LEDdb6; //DB6
P4DIR |= LEDdb6;
P4OUT &= ~LEDdb6;
uint8_t LEDdb7 = BIT7;
P4SEL1 &= ~LEDdb7;
P4SEL0 &= ~LEDdb7; //DB7
P4DIR |= LEDdb7;
P4OUT &= ~LEDdb7;
```

Here is how the LCD displays

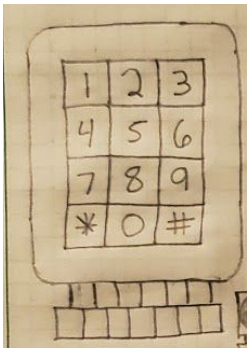
" Chase Mulder "

" Alex Kuzner "

" Slot Machine "

On the LCD display, then delays for 3 seconds:

```
//Welcome Screen
LCD_CommandWrite(0x80);
char stringType[16] = "  Chase Mulder  ";
printString(stringType);
LCD_CommandWrite(0xc0);
strcpy(stringType, "  Alex Kuzner   ");
printString(stringType);
LCD_CommandWrite(0xD0);
strcpy(stringType, "  Slot Machine  ");
printString(stringType);
SysTick_delay_ms(3000); //delay 3 seconds
```



Next, the 12 digit keypad had to be set up. Keypads are used in many devices and for this project it was “key”. The keypad is used to do many things such as maneuver the menus, determine the credit amounts and and select functions. The keypad works as a circuit so when a button is pressed it completes the circuit so that the MSP knows which one is being pressed.

The keypad circuit is all GPIO pins output to the MSP432.

The first four rows are initialized to port 2 bits 4, 5, 6, 7.

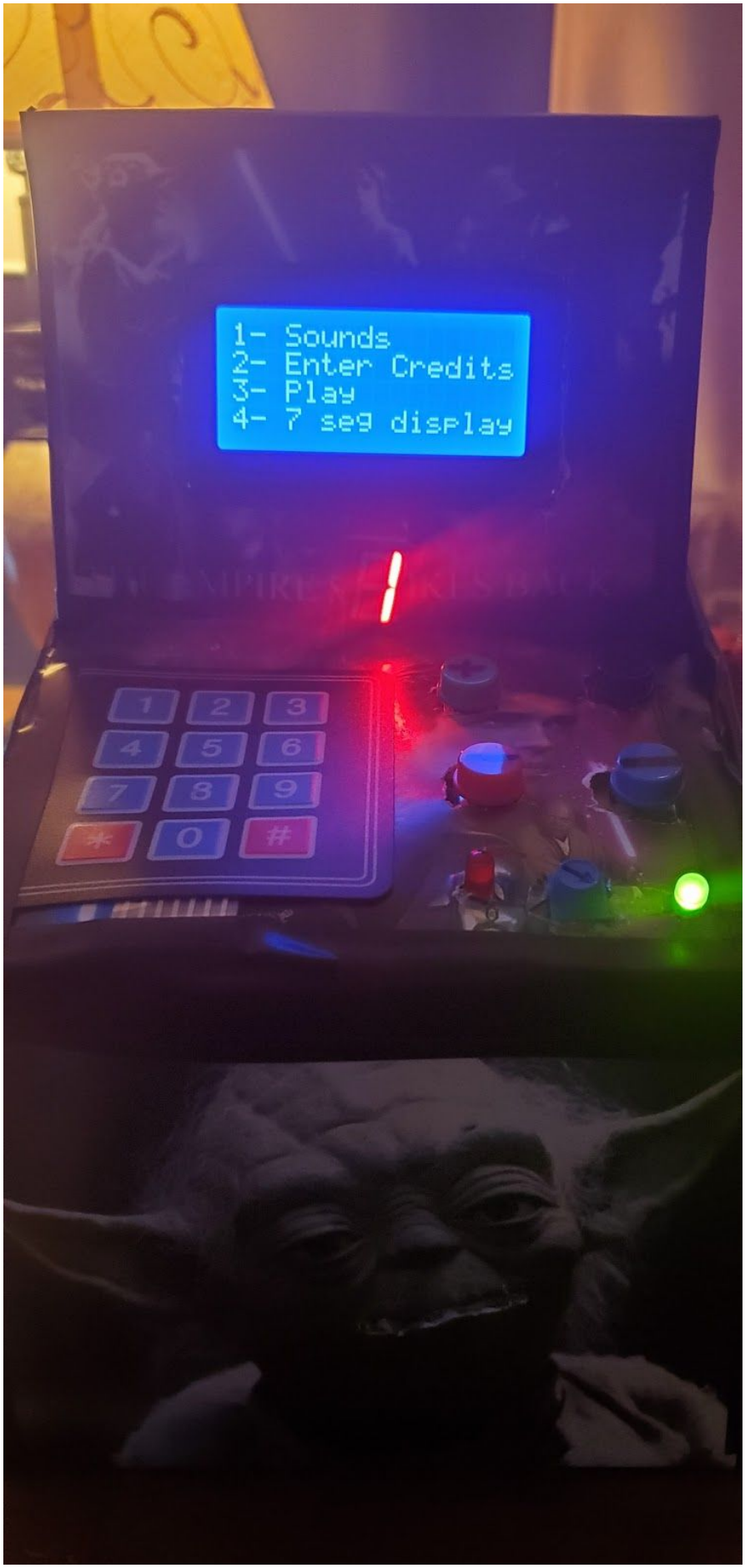
The last three columns are initialized to port 5 bits 0, 1, 2.

The most challenging part about this circuit is if you use wire extentenders and then mix up your wires somehow. Backwards wiring for the columns gave and extra challenge because it would mess up the read keypad function and would always output the wrong number because the keypad was set up wrong.

This is how the code interprets the keypress and outputs the number pressed:

```
int Read_Keypad()
{

    uint8_t row, col;
    for (col = 0; col < 3; col++)
    {
        P5->DIR &= ~(BIT0 | BIT1 | BIT2 ); //Initialize columns port 5 bits 0,1,2
        P5->DIR |= (1 << (col));
        P5->OUT &= ~(1 << (col));
        SysTick_delay_ms(10);
        row = P2->IN & 0xF0;
        while (!(P2->IN & BIT4 ) | !(P2->IN & BIT5 ) | !(P2->IN & BIT6 )
                | !(P2->IN & BIT7 ))
            ; //Initialize rows port 6 bits 0, 1, 4, 5
        if (row != 0xF0)
            break;
    }
    P5->DIR &= ~(BIT0 | BIT1 | BIT2 );
    if (col == 3)
        return 0;
    if (row == 0b11100000)
        return col + 1;
    if (row == 0b11010000)
        return 3 + col + 1;
    if (row == 0b10110000)
        return 6 + col + 1;
    if (row == 0b01110000)
        return 9 + col + 1;

    return -1;

}
```

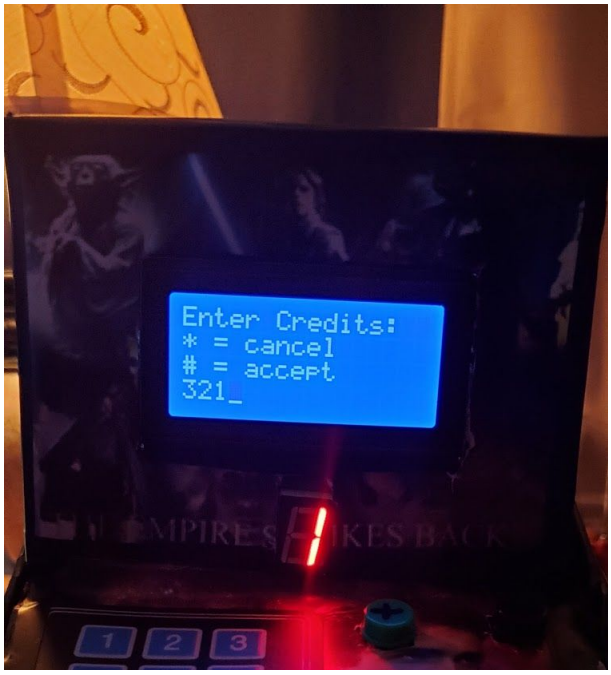
After the 3 second delay of the welcome screen, the code goes right into printing the menu screen which accepts keypresses 1-4:


1- Sounds
2- Enter Credits
3- Play
4- 7 seg display
1 2 3
4 5 6
7 8 9
* 0 #

The readkeypad function, storearray function, and printarray function all go along together and are called in the enter credits section of the menu if the user presses 2.

The function first prints its menu:

```
LCD_CommandWrite(0x80);
strcpy(stringType, "Enter Credits:  ");
printString(stringType);
LCD_CommandWrite(0xc0);
strcpy(stringType, "* = cancel      ");
printString(stringType);
LCD_CommandWrite(0x90);
strcpy(stringType, "# = accept      ");
printString(stringType);
LCD_CommandWrite(0xD0);
strcpy(stringType, "                ");
printString(stringType);
```



Next, it calls the function storeArray();. Store Array function inputs three keypresses from the user:

```
void storeArray()
{ //Bit shift left when more than 3 inputs have been inputted

   for (i = 0; i < 3;)
   {
      keypress = Read_Keypad();

      if (keypress == 1)
      {
         arraykeys[i] = keypress;
         i++;

         keypress = 13;
      }
      else if (keypress == 2)
      {
```

```
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 3)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 4)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 5)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 6)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 7)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 8)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 9)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 10)
        {

            arraykeys[0] = 0;
            arraykeys[1] = 0;
            arraykeys[2] = 0;
            i = 0;
            keypress = 13;
        }
        else if (keypress == 11)
        {
            arraykeys[i] = 0;
            i++;
        }
        else if (keypress == 12)
        {
            main_Menu();
            break;
        }

    }
    i = 0;

    for (i = 0; i < 3; i++)
    {
        if (i == 0)
        {
            newCred = arraykeys[0];
        }
        if (i == 1)
        {
            newCred = arraykeys[0] * 10;
            newCred += arraykeys[1];
        }
        if (i == 2)
        {
            newCred = arraykeys[0] * 100;
            newCred += arraykeys[1] * 10;
            newCred += arraykeys[2];
        }
```

```
    }
    print_Array();
    storeArray();
}
```

Print_Array(); then prints the users entered credit amount to the bottom left corner of the screen. Highlighted above is the for loop that calculated the 100s 10s and 1s place of the user's entered numbers and stores its total value into an integer variable called newCred.

```
void print_Array()
{
    lcdSetInt(newCred, 0, 3);
}
```

Print_Array(); prints to the LCD display at x = 0 and y = 3. After user enters credits, the only thing left to do now is listen to the sounds, update the 7 segment display, or play the game.

If the user presses "1" on the menu, then the menu advances on to the sound menu, and the sounds menu LCD prints:

```
char stringType[16] = "1- HBD          ";
LCD_CommandWrite(0x80);
strcpy(stringType, "1- HBD          ");
printString(stringType);
LCD_CommandWrite(0xc0);
strcpy(stringType, "2- STAR WARS    ");
printString(stringType);
LCD_CommandWrite(0x90);
strcpy(stringType, "3- YODA         ");
printString(stringType);
LCD_CommandWrite(0xD0);
strcpy(stringType, "                ");
printString(stringType);
LCD_CommandWrite(0xD0);
strcpy(stringType, "                ");
printString(stringType);
```

A user input of “1” will play the happy birthday song, “2” will play 8 notes to the star wars theme song, “3” will play a 2 note yoda sound. This is how the code drives the piezo buzzer:

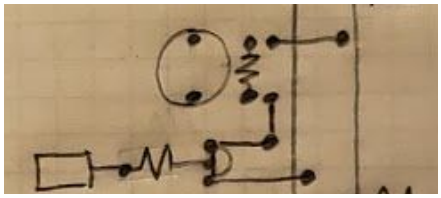
```
if (keypress == 1)
        {

            // Drive buzzer with P1.7
            P1->DIR |= BIT7;
            P1->OUT &= ~BIT7;
            P1->DIR &= ~BIT4;

            for (i = 0; i < 25; i++)
            {
                Sound_Play(8 * notes[i], 100 * interval[i]);
                pause(6);
            }
            main_Menu();
            pause(100);

        }
```

These are the notes and intervals of duration for the happy birthday song:

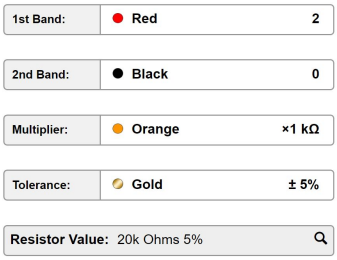
```
// Happy birthday notes
/*                  Hap py  Birth Day  to  you,  Hap py  birth day  to
 C4   C4   D4   C4   F4   E4   C4   C4   D4   C4   G4 */
unsigned int notes[] = { 262, 262, 294, 262, 349, 330, 262, 262, 294, 262, 392,
```

```
/*                    you, Hap py  Birth Day  dear  xxxx      Hap  py   birth
 F4   C4   C4   C5   A4   F4   E4   D4   B4b  B4b  A4 */
349,
                    262, 262, 523, 440, 349, 330, 294, 466, 466, 440,

                    /*                    day  to  you
                     F4   G4   F4   */
                    349,
                    392, 349 };

unsigned short interval[] = { 4, 4, 8, 8, 8, 10, 4, 4, 8, 8, 8, 10, 4, 4, 8, 8,
                          8, 8, 8, 4, 4, 8, 8, 8, 12 };
```

This is the circuit for the piezo buzzer:



The circuit uses this resistor for the piezo buzzer feedback loop and for the resistor to GPIO on port 1 bit 7:



If keypress is “4” on the menu, the 7 segment display updates its led’s and outputs the number the current contrast is at on the LCD display. This is how the 7 segment display was initialized in the code:

```
//7 segment display leds

//
P9SEL1 &= ~BIT5;
P9SEL0 &= ~BIT5;
P9DIR |= BIT5;
P9OUT &= ~BIT5;
//
P7SEL1 &= ~BIT0;
P7SEL0 &= ~BIT0;
P7DIR |= BIT0;
P7OUT &= ~BIT0;
//
P7SEL1 &= ~BIT3;
P7SEL0 &= ~BIT3;
```

```
P7DIR |= BIT3;
P7OUT &= ~BIT3;
//
P6SEL1 &= ~BIT3;
P6SEL0 &= ~BIT3;
P6DIR |= BIT3;
P6OUT &= ~BIT3;
//
P5SEL1 &= ~BIT3;
P5SEL0 &= ~BIT3;
P5DIR |= BIT3;
P5OUT &= ~BIT3;
//
P8SEL1 &= ~BIT3;
P8SEL0 &= ~BIT3;
P8DIR |= BIT3;
P8OUT &= ~BIT3;
//
P9SEL1 &= ~BIT1;
P9SEL0 &= ~BIT1;
P9DIR |= BIT1;
P9OUT &= ~BIT1;
//
P8SEL1 &= ~BIT7;
P8SEL0 &= ~BIT7;
P8DIR |= BIT7;
P8OUT &= ~BIT7;
//
P8SEL1 &= ~BIT6;
P8SEL0 &= ~BIT6;
P8DIR |= BIT6;
P8OUT &= ~BIT6;
```

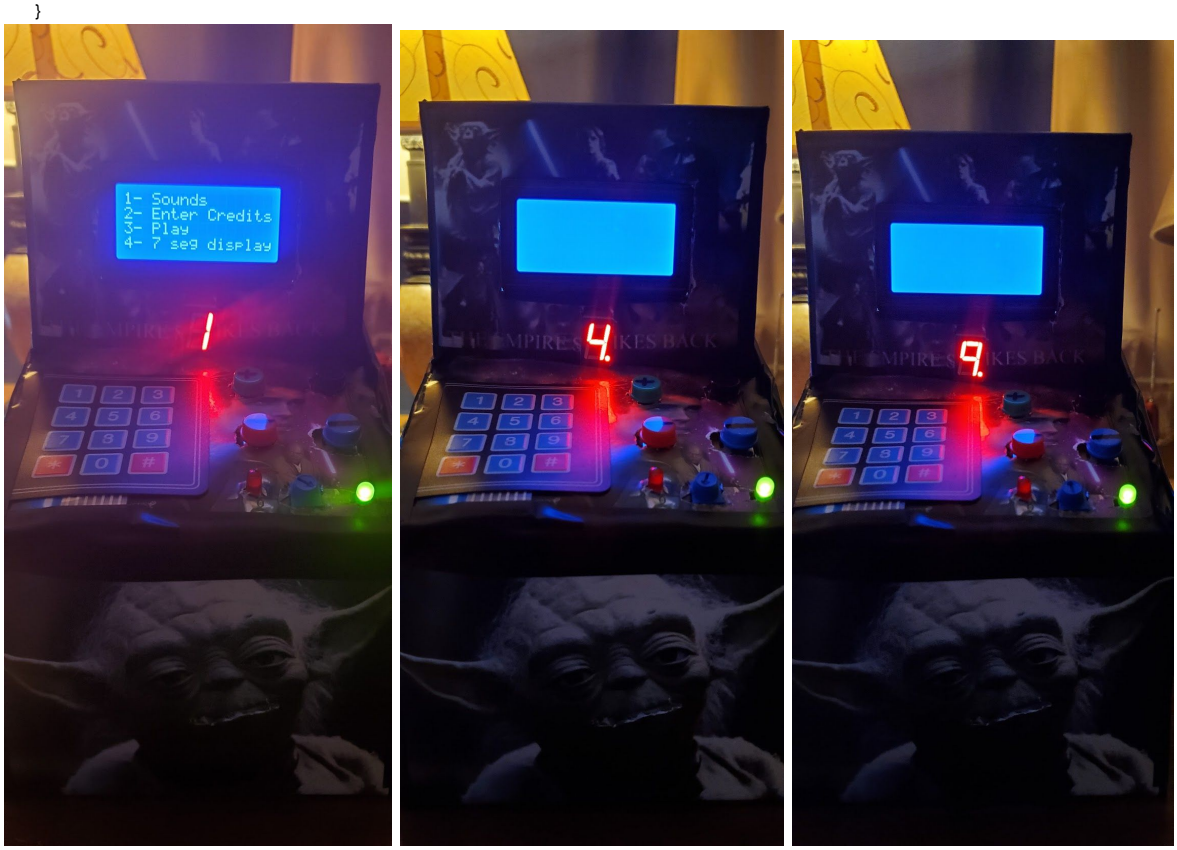
This is how the 7 segment display reads input from the 10k ohm potentiometer and outputs each LED segment for each number corresponding to a voltage change on GPIO port 5 bit 5:

```
    else if (keypress == 4) //7SEG DISPLAY
{

    //7 segment
    P7OUT = P7OUT | BIT0; //on top middle
    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT7; //on lower middle
    P8OUT = P8OUT | BIT6; //on lower left
    P8OUT = P8OUT | BIT3; //on dot
    P5OUT = P5OUT | BIT3; //on top right
    P7OUT = P7OUT | BIT3; //on top left
    P9OUT = P9OUT | BIT5; //on middle segment
    P6OUT = P6OUT | BIT3; //on top middle
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////
    uint32_t i = 0;
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;   // stop watchdog timer

    //7 Segment display
    ////////////////////////////////////
    float nADC;
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
    ADC14_pinInit();
    ADC14_preiphInit();

    uint16_t result;
    ADC14->CTL0 |= ADC14_CTL0_SC;
    while (!ADC14->IFGR0 & BIT0 )
        ;
    result = ADC14->MEM[0];
```

```
nADC = (result * 3.3) / 16384;

printf("Value is:\n\t%d\n", result);
SysTick_delay_ms(100);

if (result < 2000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////
    P9OUT = P9OUT | BIT1; //on lower right
    P5OUT = P5OUT | BIT3; //on top right

}
if (result < 3000 && result > 2000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////
    P7OUT = P7OUT | BIT0; //on top middle
    P5OUT = P5OUT | BIT3; //on top right
    P9OUT = P9OUT | BIT5; //on middle segment
    P8OUT = P8OUT | BIT6; //on lower left
    P8OUT = P8OUT | BIT7; //on lower middle
}
if (result < 4000 && result > 3000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////
    P7OUT = P7OUT | BIT0; //on top middle
    P5OUT = P5OUT | BIT3; //on top right
    P9OUT = P9OUT | BIT5; //on middle segment
    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT7; //on lower middle
}
if (result < 5000 && result > 4000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////

    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT3; //on dot
    P5OUT = P5OUT | BIT3; //on top right
    P7OUT = P7OUT | BIT3; //on top left
    P9OUT = P9OUT | BIT5; //on middle segment

}
if (result < 6000 && result > 5000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
```

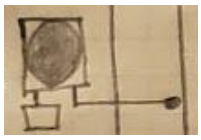
```
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////
    P7OUT = P7OUT | BIT0; //on top middle
    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT7; //on lower middle
    P8OUT = P8OUT | BIT3; //on dot
    P7OUT = P7OUT | BIT3; //on top left
    P9OUT = P9OUT | BIT5; //on middle segment
    P6OUT = P6OUT | BIT3; //on top middle
}
if (result < 7000 && result > 6000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////

    P7OUT = P7OUT | BIT0; //on top middle
    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT7; //on lower middle
    P8OUT = P8OUT | BIT6; //on lower left
    P8OUT = P8OUT | BIT3; //on dot
    P9OUT = P9OUT | BIT5; //on middle segment
    P7OUT = P7OUT | BIT3; //on top left
    P6OUT = P6OUT | BIT3; //on top middle

}
if (result < 8000 && result > 7000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////
    P7OUT = P7OUT | BIT0; //on top middle
    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT3; //on dot
    P5OUT = P5OUT | BIT3; //on top right

}
if (result < 9000 && result > 8000)
{
    ////////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////////
    P7OUT = P7OUT | BIT0; //on top middle
    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT7; //on lower middle
    P8OUT = P8OUT | BIT6; //on lower left
    P8OUT = P8OUT | BIT3; //on dot
    P5OUT = P5OUT | BIT3; //on top right
    P7OUT = P7OUT | BIT3; //on top left
    P9OUT = P9OUT | BIT5; //on middle segment
    P6OUT = P6OUT | BIT3; //on top middle

}
if (result < 12000 && result > 9000)
{
    ////////////////////////////////////
```

```
        P7OUT = P7OUT & ~BIT0; //off
        P9OUT = P9OUT & ~BIT1; //off
        P8OUT = P8OUT & ~BIT7; //off
        P8OUT = P8OUT & ~BIT6; //off
        P8OUT = P8OUT & ~BIT3; //off
        P5OUT = P5OUT & ~BIT3; //off
        P7OUT = P7OUT & ~BIT3; //off
        P9OUT = P9OUT & ~BIT5; //off
        P6OUT = P6OUT & ~BIT3; //off
        ////////////////////////////////
        P7OUT = P7OUT | BIT0; //on top middle
        P9OUT = P9OUT | BIT1; //on lower right
        P8OUT = P8OUT | BIT3; //on dot
        P5OUT = P5OUT | BIT3; //on top right
        P7OUT = P7OUT | BIT3; //on top left
        P9OUT = P9OUT | BIT5; //on middle segment
        P6OUT = P6OUT | BIT3; //on top middle
    }
    ////////////////////////////////////////
}
```



Last of all, if user enters “3” on the menu, then the code starts to play the game. The play game function reads inputs from the button and the IR sensor and interprets what button is being pressed and what to do. Button 1 is the increase bet button and it increases the bet by one each time the user pressed the button, or holds the button for

a second. Button 2 is the decrease bet button and does the opposite of increase bet, but decreases the bet by 1. This is how the buttons are wired:

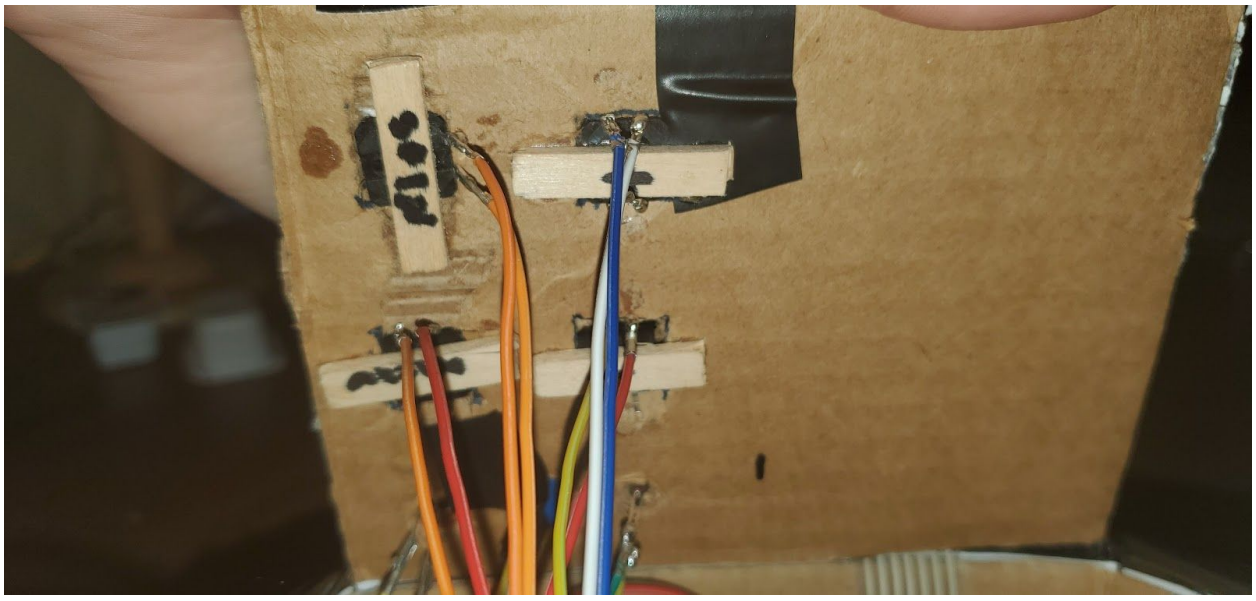


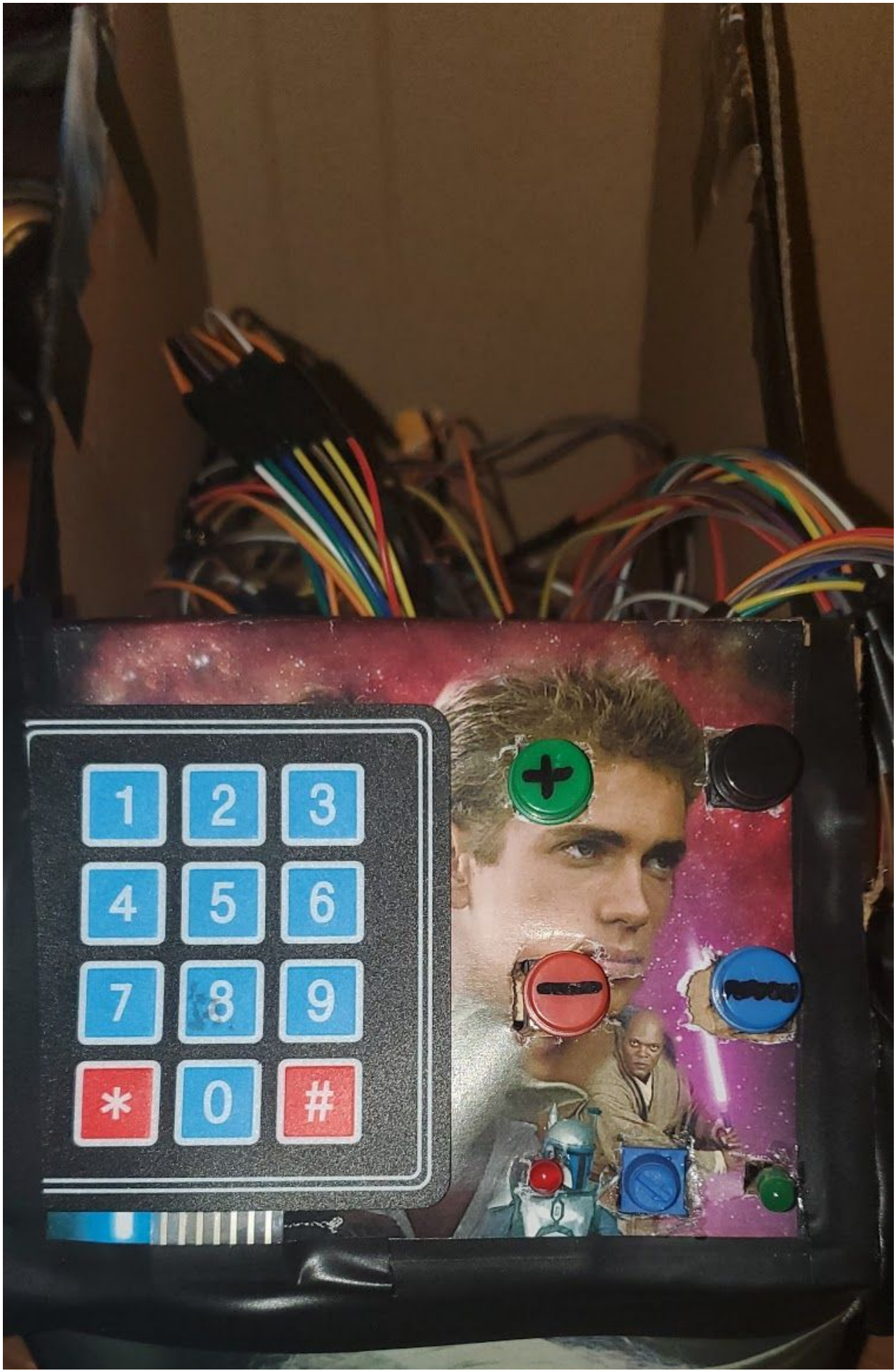
This is how the buttons are initialized:

```
//Initializing Button 1
P6DIR &= ~BIT0;
P6REN |= BIT0;
P6OUT |= BIT0;

//Initializing Button 2
P6DIR &= ~BIT1;
P6REN |= BIT1;
P6OUT |= BIT1;
//Initializing Button 3
P3DIR &= ~BIT2;
P3REN |= BIT2;
P3OUT |= BIT2;
//Initializing Button 4
P3DIR &= ~BIT3;
P3REN |= BIT3;
P3OUT |= BIT3;
```

This is how the code returns the status of the button:

```
char SwitchStatus_Launchpad_Button1()

{
   return (P6IN & BIT0 );
}
```

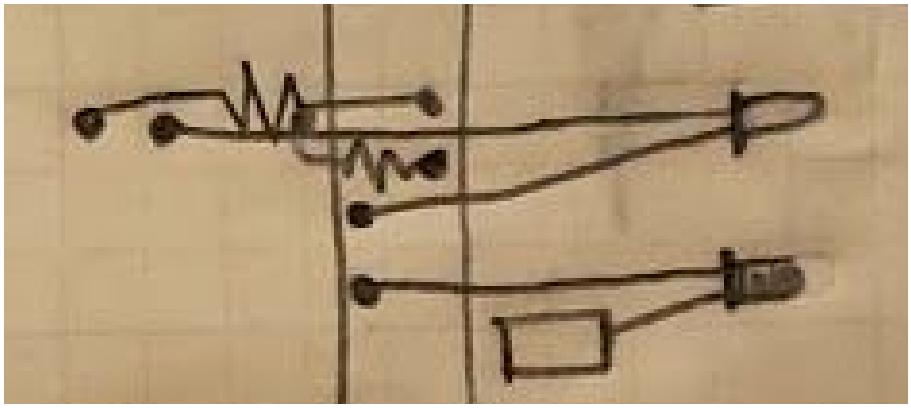


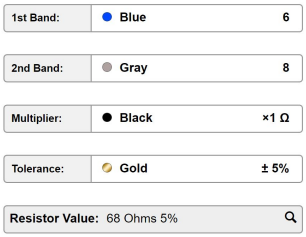

1 2 3
4 5 6
7 8 9
* 0 #

The IR sensor checks if the IR beam is broken, and if it is it adds 1 to bet and plays a little noise. This is how the IR sensor works in the code on port 4 bit 3:

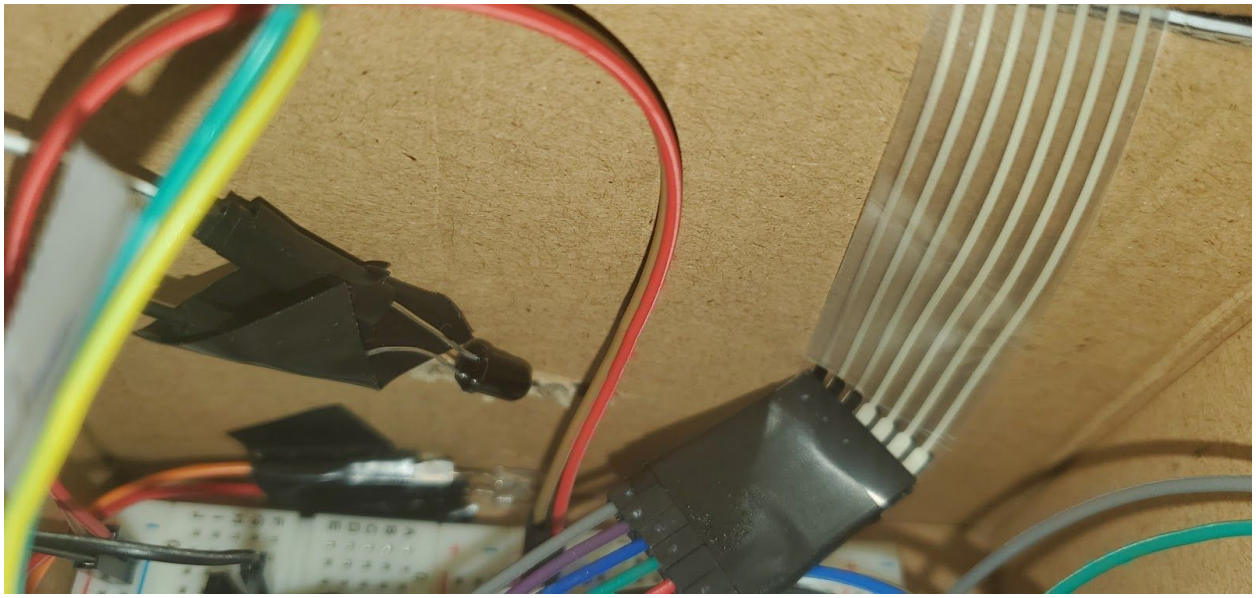
```
if (P4IN & BIT3)
 {
    printf("working coin\n");

    newCred = newCred + 1;
    printf("working\n");
    P1->DIR |= BIT7;
    P1->OUT &= ~BIT7;
    P1->DIR &= ~BIT4;

    for (i = 0; i < 2; i++)
    {
       Sound_Play(8 * notes3[i], 100 * interval3[i]);
       pause(6);
    }

    button_status = 0;
    P4IN &= ~BIT3;
    print_Array();
    playGame();

 }
```

This is the circuit for the IR sensor:





Use two 68 ohm resistors.



Button “3” is used to go back to the menu if the user decides they want to change their credit amount, or listen to another sound, or update the 7 segment display.

Button “4” is the Spin function button. The spin function is where the actual slot machine game is played. Credits and bet are first printed in their respective positions at the bottom of each corner:

```
print_Array(); //Credits in lower left
lcdSetInt(credits_array[0], 13, 3);
lcdSetInt(credits_array[1], 14, 3);
lcdSetInt(credits_array[2], 15, 3); //Bet in lower right
```

Next, ten random numbers between 0 and 2 are stored into and integer array of size 8 using this function:

```
t[0] = rand() % 3;
```

Next, an array of characters is then assigned its character in a for loop. A stick figure = 1, a bell = 2, and a smile = 3:

```
sprintf(r[8], "%d%d%d%d%d%d%d%d", t[0], t[1], t[2], t[3], t[4], t[5], t[6],
```

```
        t[7]);

for (i = 0; i < 8; i++)
{
    if (t[i] == 0)
    {
        r[i] = stickfig_index;
    }
    else if (t[i] == 1)
    {
        r[i] = bell_index;
    }
    else
    {
        r[i] = smile_index;
    }

}
```

Next, an animation plays and the 8 random characters are printed to the LCD screen:

```
////////////////////////////////////////Animation

lcdSetChar(stickfig_index, 1, 0);
lcdSetChar(bell_index, 2, 0);
lcdSetChar(smile_index, 3, 0);
lcdSetChar(stickfig_index, 4, 0);
lcdSetChar(bell_index, 1, 1);
lcdSetChar(smile_index, 2, 1);
lcdSetChar(stickfig_index, 3, 1);
lcdSetChar(bell_index, 4, 1);
SysTick_delay_ms(150);
//
lcdSetChar(bell_index, 1, 0);
lcdSetChar(smile_index, 2, 0);
lcdSetChar(stickfig_index, 3, 0);
lcdSetChar(bell_index, 4, 0);
lcdSetChar(smile_index, 1, 1);
lcdSetChar(stickfig_index, 2, 1);
lcdSetChar(bell_index, 3, 1);
lcdSetChar(smile_index, 4, 1);
SysTick_delay_ms(150);
//
lcdSetChar(smile_index, 1, 0);
lcdSetChar(bell_index, 2, 0);
lcdSetChar(stickfig_index, 3, 0);
lcdSetChar(smile_index, 4, 0);
lcdSetChar(bell_index, 1, 1);
lcdSetChar(stickfig_index, 2, 1);
lcdSetChar(smile_index, 3, 1);
lcdSetChar(bell_index, 4, 1);
SysTick_delay_ms(150);
////////////////////////////////////////////////

lcdSetChar(r[0], 1, 0);
lcdSetChar(r[1], 2, 0);
lcdSetChar(r[2], 3, 0);
lcdSetChar(r[3], 4, 0);
lcdSetChar(r[4], 1, 1);
lcdSetChar(r[5], 2, 1);
lcdSetChar(r[6], 3, 1);
lcdSetChar(r[7], 4, 1);
```

Finally, win/loss conditions are assessed and amount won, or lost is printed to the screen above the bets and the total credit amount is updated:

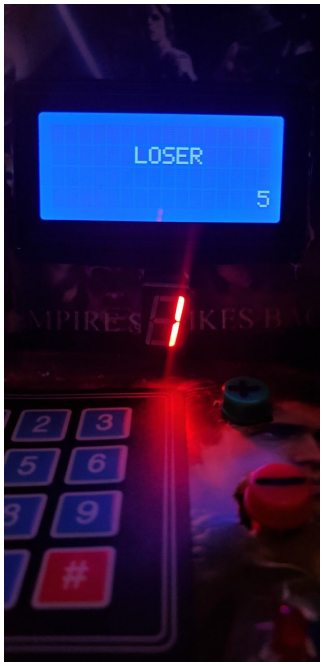
```
//win conditions

for (i = 0; i < 8; i++)
{
    if (t[i] == 0 || t[i] == 2)
    {
        totalCredits += 1 * credits_array[2]; //stickfig  //smile
    }
    else
    {
        totalCredits = -1 * credits_array[2]; //bell
    }
}
```

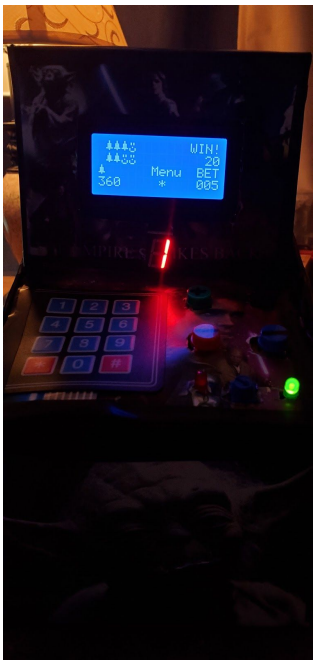
```
lcdSetInt(totalCredits, 14, 1); //print amount won or lost

newCred = newCred + totalCredits;
print_Array(); //print amount won or lost
```

The game Star Slots simply adds a point multiplied by your bet if you get a smile, or stick figure, but it subtracts a point multiplied by your bet for each bell you get.

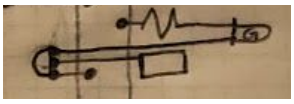
Last, the game will print “WIN!” if your total credits amount won is greater than 0, or will print “LOSER” with a short delay if total credits won is less than 0:

```
if (totalCredits > 0)
{
   LCD_CommandWrite(0x86);
   strcpy(stringGame, "      WIN!");
   printString(stringGame);
   TurnOn_Green_LED();

   printf("Win %d", totalCredits);

}

if (totalCredits < 0)
{
   LCD_CommandWrite(0x86);
   strcpy(stringGame, "     LOSE!");
   printString(stringGame);

   TurnOn_Red_LED();
   printf("Lose %d", totalCredits);

   {
      char stringType[16] = "                ";
      LCD_CommandWrite(0x80);
      strcpy(stringType, "                ");
      printString(stringType);
      LCD_CommandWrite(0xc0);
      strcpy(stringType, "      LOSER     ");
      printString(stringType);
      LCD_CommandWrite(0x90);
      strcpy(stringType, "                ");
      printString(stringType);
      LCD_CommandWrite(0xD0);
      strcpy(stringType, "                ");
      printString(stringType);

   }

}
```





If the user wins, the green led turns on and stays on; if the user loses, the red led turns on and stays on.



The green led uses a 70 ohm resistor.
The red led uses a 120 ohm resistor.

Here's how the led's are initialized:

```
//Initializing green LED
uint8_t greenLED = BIT6;
P3SEL1 &= ~greenLED;
P3SEL0 &= ~greenLED; //Green
P3DIR |= greenLED;
```

Here's the functions to turn on, or off the led's:

```
void TurnOn_Green_LED()
{
  P3OUT |= P3OUT | BIT6;
}
void TurnOff_Green_LED()
{
  P6OUT = P6OUT & ~BIT6;
}
```

The game then waits for button input from the user. If the menu button is pressed, a "GAME OVER" screen is displayed with the user's current amount of credits at the bottom left of the screen.


HOPE YA HAD FUN!
GAME OVER
CREDITS
340
1 2 3
4 5 6
7 8 9
* 0 #

Results:

```
//*****************************************************************************************************
//*Name:              Chase Mulder and Alex Kuzner
//*Course            EGR226 section 901 professor Trevor Ekins
//*Project           STAR SLOTS FINAL REPORT
//*File              STARSLOTS.c
//*Description       Final EGR 226 Project - PLAY OUR GAME:)
//*****************************************************************************************************

//////LIBRARIES////////////
#include <driverlib.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "LCD_Library.h"
#include "SysTick_Library.h"
#include "LCD_Custom.h"
#include "led_source.h"
#include "seg_source.h"
#include <seg_source.h>
#include "button_source.h"
#include "lcd_source.h"
#include "ir_source.h"
#include "keypad_source.h"
//////////////////////////

////FUNCTIONS////////////////////////////////
void led_source();
void seg_source();
void button_source();
void lcd_source();
void ir_source();
void keypad_source();
void initialize();
void TurnOn_Green_LED();
void TurnOff_Green_LED();
void TurnOn_Red_LED();
void TurnOff_Red_LED();
char SwitchStatus_Launchpad_Button1();
char SwitchStatus_Launchpad_Button2();
char SwitchStatus_Launchpad_Button3();
char SwitchStatus_Launchpad_Button4();
void LCD_Init(void);
void LCD_PulseEnablePin(void);
void LCD_PushNibble(uint8_t nibble);
void LCD_PushByte(uint8_t byte);
void LCD_CommandWrite(uint8_t command);
void LCD_DataWrite(uint8_t data);
void printString(char stringType[]);
void TimerA2Init();
void TAO_N_IRQHandler();
void finalArray();
void storeArray();
int Read_Keypad();
void print_Array();
void Sound_Play(unsigned, unsigned);
void pause(unsigned short);
void main_Menu();
void playGame();
void Spin();
//////////////////////////////////////////

/////VARIABLES////////////////////////////////
#define PRESSED 0
volatile int newCred = 0;
int button_status = 0;
unsigned short lastedge, currentedge, period;
int newFlag;
int keypress;
int arraykeys[3];
int credits_array[3];
int i = 0;
char bell_index;
char stickfig_index;
char smile_index;
//////////////////////////////////////////

//////SONGS//////////////////////////////////////////////////////////////////////////////
// Happy birthday notes
/*                  Hap py  Birth Day  to  you,  Hap py  birth day  to
 C4   C4   D4   C4   F4   E4   C4   C4   D4   C4   G4 */
unsigned int notes[] = { 262, 262, 294, 262, 349, 330, 262, 262, 294, 262, 392,

/*                  you, Hap py  Birth Day  dear  xxxx      Hap  py   birth
 F4   C4   C4   C5   A4   F4   E4   D4   B4b  B4b  A4 */
349,
                    262, 262, 523, 440, 349, 330, 294, 466, 466, 440,

                    /*                  day  to  you
                     F4   G4   F4   */
                    349,
                    392, 349 };

unsigned short interval[] = { 4, 4, 8, 8, 8, 10, 4, 4, 8, 8, 8, 10, 4, 4, 8, 8,
                    8, 8, 8, 4, 4, 8, 8, 8, 12 };
#define c 261
#define d 294
#define e 329
#define f 349
#define g 391
#define gS 415
#define a 440
#define aS 455
#define b 466
#define cH 523
#define cSH 554
#define dH 587
#define dSH 622
#define eH 659
#define fH 698
#define fSH 740
#define gH 784
#define gSH 830
#define aH 880
// STAR WARS
unsigned int notes2[] = { a, a, a, f, cH, a, f, cH, a };
unsigned short interval2[] = { 4, 4, 8, 8, 8, 10, 4, 4, };

//Yoda noise
unsigned int notes3[] = { a, f };
unsigned short interval3[] = { 20, 20 };
//////////////////////////////////////////////////////////////////////////////////////////

//////////SPIN FUNCTION THAT PLAYS THE GAME////////////////////
void Spin()
{

    print_Array(); //Credits in lower left
    lcdSetInt(credits_array[0], 13, 3);
    lcdSetInt(credits_array[1], 14, 3);
    lcdSetInt(credits_array[2], 15, 3); //Bet in lower right

    TurnOff_Green_LED();
    TurnOff_Red_LED();
```

```
char stringGame[16] = "                ";
LCD_CommandWrite(0x90);
strcpy(stringGame, "CR     Menu  BET");
printString(stringGame);
LCD_CommandWrite(0xD0);
strcpy(stringGame, "        *       ");
printString(stringGame);

bell_index = lcdCreateCustomChar(&bell_layout);
stickfig_index = lcdCreateCustomChar(&stickfig_layout);
smile_index = lcdCreateCustomChar(&smile_layout);

lcdSetChar(bell_index, 0, 0);
lcdSetChar(stickfig_index, 1, 0);
lcdSetChar(smile_index, 2, 0);

lcdInit();

char r[8];
int t[8];

printf("Ten random numbers in [1,3]\n");

t[0] = rand() % 3;
t[1] = rand() % 3;
t[2] = rand() % 3;
t[3] = rand() % 3;
t[4] = rand() % 3;
t[5] = rand() % 3;
t[6] = rand() % 3;
t[7] = rand() % 3;

sprintf(r[8], "%d%d%d%d%d%d%d%d", t[0], t[1], t[2], t[3], t[4], t[5], t[6],
        t[7]);

for (i = 0; i < 8; i++)
{
    if (t[i] == 0)
    {
        r[i] = stickfig_index;
    }
    else if (t[i] == 1)
    {
        r[i] = bell_index;
    }
    else
    {
        r[i] = smile_index;
    }

}

////////////////////////////////////////Animation
lcdSetChar(stickfig_index, 1, 0);
lcdSetChar(bell_index, 2, 0);
lcdSetChar(smile_index, 3, 0);
lcdSetChar(stickfig_index, 4, 0);
lcdSetChar(bell_index, 1, 1);
lcdSetChar(smile_index, 2, 1);
lcdSetChar(stickfig_index, 3, 1);
lcdSetChar(bell_index, 4, 1);
SysTick_delay_ms(150);
//
lcdSetChar(bell_index, 1, 0);
lcdSetChar(smile_index, 2, 0);
lcdSetChar(stickfig_index, 3, 0);
lcdSetChar(bell_index, 4, 0);
lcdSetChar(smile_index, 1, 1);
lcdSetChar(stickfig_index, 2, 1);
lcdSetChar(bell_index, 3, 1);
lcdSetChar(smile_index, 4, 1);
SysTick_delay_ms(150);
//
lcdSetChar(smile_index, 1, 0);
lcdSetChar(bell_index, 2, 0);
lcdSetChar(stickfig_index, 3, 0);
lcdSetChar(smile_index, 4, 0);
lcdSetChar(bell_index, 1, 1);
lcdSetChar(stickfig_index, 2, 1);
lcdSetChar(smile_index, 3, 1);
lcdSetChar(bell_index, 4, 1);
SysTick_delay_ms(150);
////////////////////////////////////////////////

lcdSetChar(r[0], 1, 0);
lcdSetChar(r[1], 2, 0);
lcdSetChar(r[2], 3, 0);
lcdSetChar(r[3], 4, 0);
lcdSetChar(r[4], 1, 1);
lcdSetChar(r[5], 2, 1);
lcdSetChar(r[6], 3, 1);
lcdSetChar(r[7], 4, 1);

stringGame[16] = "                ";
LCD_CommandWrite(0x90);
strcpy(stringGame, "CR     Menu  BET");
printString(stringGame);
LCD_CommandWrite(0xD0);
strcpy(stringGame, "        *       ");
printString(stringGame);
print_Array(); //Credits in lower left
lcdSetInt(credits_array[0], 13, 3);
lcdSetInt(credits_array[1], 14, 3);
lcdSetInt(credits_array[2], 15, 3); //Bet in lower right

int updatedCredits[3];
int totalCredits = 0;
//win conditions

for (i = 0; i < 8; i++)
{
    if (t[i] == 0 || t[i] == 2)
    {
        totalCredits += 1 * credits_array[2]; //stickfig  //smile
    }
    else
    {
        totalCredits = -1 * credits_array[2]; //bell
    }
}
lcdSetInt(totalCredits, 14, 1); //print amount won or lost

newCred = newCred + totalCredits;
print_Array(); //print amount won or lost

if (totalCredits > 0)
{
    LCD_CommandWrite(0x86);
    strcpy(stringGame, "      WIN!");
    printString(stringGame);
    TurnOn_Green_LED();

    printf("Win %d", totalCredits);

}

if (totalCredits < 0)
{
    LCD_CommandWrite(0x86);
    strcpy(stringGame, "     LOSE!");
    printString(stringGame);

    TurnOn_Red_LED();
    printf("Lose %d", totalCredits);
```

```
    {
        char stringType[16] = "                ";
        LCD_CommandWrite(0x80);
        strcpy(stringType, "                ");
        printString(stringType);
        LCD_CommandWrite(0xc0);
        strcpy(stringType, "     LOSER     ");
        printString(stringType);
        LCD_CommandWrite(0x90);
        strcpy(stringType, "                ");
        printString(stringType);
        LCD_CommandWrite(0xD0);
        strcpy(stringType, "                ");
        printString(stringType);

    }

}

totalCredits = 0;

lcdSetInt(credits_array[2], 15, 3); //Bet in lower right

////////////////////////////////////////////////////////////CHECK BUTTONS/////////
if (SwitchStatus_Launchpad_Button1() == PRESSED)
{ //Button debouncing
    SysTick_delay_ms(3);
    if (SwitchStatus_Launchpad_Button1() == PRESSED)
    {
        button_status = 1;
    }
}

if (button_status == 1)
{
    printf("work\n");
    if (credits_array[2] < arraykeys[0] * 100
            & credits_array[2] < arraykeys[1] * 10 & credits_array[2] < 5)
    {
        credits_array[2] = credits_array[2] + 1;
        printf("working\n");
        button_status = 0;
        SysTick_delay_ms(10);

    }

    button_status = 0;

}
////////////////////////////////////////////////////////////
if (SwitchStatus_Launchpad_Button2() == PRESSED)
{ //Button debouncing
    SysTick_delay_ms(3);
    if (SwitchStatus_Launchpad_Button2() == PRESSED)
    {
        button_status = 1;
    }
}

if (button_status == 1)
{
    printf("work\n");
    if (credits_array[2] > 0)
    {
        credits_array[2] = credits_array[2] - 1;
        printf("working\n");
        button_status = 0;
        SysTick_delay_ms(10);

    }

    button_status = 0;
}

////////////////////////////////////////////////////////////
if (SwitchStatus_Launchpad_Button3() == PRESSED)
{
    char stringType[16] = "                ";
    LCD_CommandWrite(0x80);
    strcpy(stringType, "                ");
    printString(stringType);
    LCD_CommandWrite(0xc0);
    strcpy(stringType, "  GAME OVER     ");
    printString(stringType);
    LCD_CommandWrite(0x90);
    strcpy(stringType, "                ");
    printString(stringType);
    LCD_CommandWrite(0xD0);
    strcpy(stringType, "                ");
    printString(stringType);
    print_Array();
    SysTick_delay_ms(300000000000000);

    main_Menu();
}
////////////////////////////////////////////////////////////
if (SwitchStatus_Launchpad_Button4() == PRESSED)
{ //Button debouncing
    SysTick_delay_ms(3);
    if (SwitchStatus_Launchpad_Button4() == PRESSED)
    {
        button_status = 1;
    }
}

if (button_status == 1)
{
    button_status = 0;
    Spin();
}
////////////////////////////////////////////////////////////

while (SwitchStatus_Launchpad_Button4() != PRESSED)
{
    if (SwitchStatus_Launchpad_Button4() == PRESSED)
    {
        Spin();
    }

    if (SwitchStatus_Launchpad_Button3() == PRESSED)
    { //Button debouncing
        SysTick_delay_ms(3);
        if (SwitchStatus_Launchpad_Button3() == PRESSED)
        {
            button_status = 1;
        }
    }

    if (button_status == 1)
    {
        char stringType[16] = "HOPE YA HAD FUN!";
        LCD_CommandWrite(0x80);
        strcpy(stringType, "HOPE YA HAD FUN!");
        printString(stringType);
        LCD_CommandWrite(0xc0);
        strcpy(stringType, "GAME OVER       ");
        printString(stringType);
        LCD_CommandWrite(0x90);
        strcpy(stringType, "CREDITS         ");
        printString(stringType);
        LCD_CommandWrite(0xD0);
        strcpy(stringType, "                ");
        printString(stringType);
```

```
            print_Array();
            SysTick_delay_ms(300000000000000);

            main_Menu();

            button_status = 0;

        }
        ////////////////////////////////////////////////////////
        if (SwitchStatus_Launchpad_Button2() == PRESSED)
        { //Button debouncing
            SysTick_delay_ms(3);
            if (SwitchStatus_Launchpad_Button2() == PRESSED)
            {
                button_status = 1;
            }
        }

        if (button_status == 1)
        {
            printf("work\n");
            if (credits_array[2] > 0)
            {
                arraykeys[2] = credits_array[2] - 1;
                printf("working\n");
                button_status = 0;
                SysTick_delay_ms(10);

            }

            button_status = 0;
        }
    }
}

////////////////////////////////////////////////MAIN//
int main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;

    //SysTick_Init();
    SysTickInit_NoInterrupts();
    initialize();
    lcdInit();

    credits_array[0] = 0;
    credits_array[1] = 0;
    credits_array[2] = 0;

    //Welcome Screen
    LCD_CommandWrite(0x80);
    char stringType[16] = "  Chase Mulder  ";
    printString(stringType);
    LCD_CommandWrite(0xc0);
    strcpy(stringType, "  Alex Kuzner   ");
    printString(stringType);
    LCD_CommandWrite(0xD0);
    strcpy(stringType, "  Slot Machine  ");
    printString(stringType);
    SysTick_delay_ms(3000); //delay 3 seconds

    //Main menu function
    main_Menu();

} //end of main

void main_Menu()
{
    while (1)
    {
        //Main Menu
        char stringType[16] = "  Chase Mulder  ";
        LCD_CommandWrite(0x80);
        strcpy(stringType, "1- Sounds       ");
        printString(stringType);
        LCD_CommandWrite(0xc0);
        strcpy(stringType, "2- Enter Credits");
        printString(stringType);
        LCD_CommandWrite(0x90);
        strcpy(stringType, "3- Play         ");
        printString(stringType);
        LCD_CommandWrite(0xD0);
        strcpy(stringType, "                ");
        printString(stringType);
        LCD_CommandWrite(0xD0);
        strcpy(stringType, "4- 7 seg display");
        printString(stringType);

        keypress = Read_Keypad();

        if (keypress == 1) //SOUNDS
        {
            while (SwitchStatus_Launchpad_Button3() != PRESSED)
            {

                keypress = Read_Keypad();

                char stringType[16] = "1- HBD          ";
                LCD_CommandWrite(0x80);
                strcpy(stringType, "1- HBD          ");
                printString(stringType);
                LCD_CommandWrite(0xc0);
                strcpy(stringType, "2- STAR WARS    ");
                printString(stringType);
                LCD_CommandWrite(0x90);
                strcpy(stringType, "3- YODA         ");
                printString(stringType);
                LCD_CommandWrite(0xD0);
                strcpy(stringType, "                ");
                printString(stringType);
                LCD_CommandWrite(0xD0);
                strcpy(stringType, "                ");
                printString(stringType);

                if (keypress == 1)
                {

                    // Drive buzzer with P1.7
                    P1->DIR |= BIT7;
                    P1->OUT &= ~BIT7;
                    P1->DIR &= ~BIT4;

                    for (i = 0; i < 25; i++)
                    {
                        Sound_Play(8 * notes[i], 100 * interval[i]);
                        pause(6);
                    }
                    main_Menu();
                    pause(100);

                }

                else if (keypress == 2)
                {

                    keypress = Read_Keypad();

                    // Drive buzzer with P1.7
                    P1->DIR |= BIT7;
                    P1->OUT &= ~BIT7;
                    P1->DIR &= ~BIT4;

                    for (i = 0; i < 25; i++)
```

```
            {
                Sound_Play(8 * notes2[i], 100 * interval2[i]);
                pause(6);
            }
            main_Menu();
            pause(100);

        }

        else if (keypress == 3)
        {

        }

        //
    } //end while
}

else if (keypress == 2) //CREDITS
{
    //Enter credits

    LCD_CommandWrite(0x80);
    strcpy(stringType, "Enter Credits:  ");
    printString(stringType);
    LCD_CommandWrite(0xc0);
    strcpy(stringType, "* = cancel      ");
    printString(stringType);
    LCD_CommandWrite(0x90);
    strcpy(stringType, "# = accept      ");
    printString(stringType);
    LCD_CommandWrite(0xD0);
    strcpy(stringType, "                ");
    printString(stringType);

    storeArray(); //Credit storing

}
else if (keypress == 3) //PLAY GAME
{
    char stringGame[16] = "                ";
    LCD_CommandWrite(0x80);
    strcpy(stringGame, "                ");
    printString(stringGame);
    LCD_CommandWrite(0xc0);
    strcpy(stringGame, "                ");
    printString(stringGame);
    playGame();
}

else if (keypress == 4) //7SEG DISPLAY
{

    //7 segment
    P7OUT = P7OUT | BIT0; //on top middle
    P9OUT = P9OUT | BIT1; //on lower right
    P8OUT = P8OUT | BIT7; //on lower middle
    P8OUT = P8OUT | BIT6; //on lower left
    P8OUT = P8OUT | BIT3; //on dot
    P5OUT = P5OUT | BIT3; //on top right
    P7OUT = P7OUT | BIT3; //on top left
    P9OUT = P9OUT | BIT5; //on middle segment
    P6OUT = P6OUT | BIT3; //on top middle
    ////////////////////////////////
    P7OUT = P7OUT & ~BIT0; //off
    P9OUT = P9OUT & ~BIT1; //off
    P8OUT = P8OUT & ~BIT7; //off
    P8OUT = P8OUT & ~BIT6; //off
    P8OUT = P8OUT & ~BIT3; //off
    P5OUT = P5OUT & ~BIT3; //off
    P7OUT = P7OUT & ~BIT3; //off
    P9OUT = P9OUT & ~BIT5; //off
    P6OUT = P6OUT & ~BIT3; //off
    ////////////////////////////////
    uint32_t i = 0;
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;   // stop watchdog timer

    //7 Segment display
    ////////////////////////////////
    float nADC;
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
    ADC14_pinInit();
    ADC14_preiphInit();

    uint16_t result;
    ADC14->CTL0 |= ADC14_CTL0_SC;
    while (!ADC14->IFGR0 & BIT0 )
        ;
    result = ADC14->MEM[0];
    nADC = (result * 3.3) / 16384;

    printf("Value is:\n\t%d\n", result);
    SysTick_delay_ms(100);

    if (result < 2000)
    {
        ////////////////////////////////
        P7OUT = P7OUT & ~BIT0; //off
        P9OUT = P9OUT & ~BIT1; //off
        P8OUT = P8OUT & ~BIT7; //off
        P8OUT = P8OUT & ~BIT6; //off
        P8OUT = P8OUT & ~BIT3; //off
        P5OUT = P5OUT & ~BIT3; //off
        P7OUT = P7OUT & ~BIT3; //off
        P9OUT = P9OUT & ~BIT5; //off
        P6OUT = P6OUT & ~BIT3; //off
        ////////////////////////////////
        P9OUT = P9OUT | BIT1; //on lower right
        P5OUT = P5OUT | BIT3; //on top right

    }
    if (result < 3000 && result > 2000)
    {
        ////////////////////////////////
        P7OUT = P7OUT & ~BIT0; //off
        P9OUT = P9OUT & ~BIT1; //off
        P8OUT = P8OUT & ~BIT7; //off
        P8OUT = P8OUT & ~BIT6; //off
        P8OUT = P8OUT & ~BIT3; //off
        P5OUT = P5OUT & ~BIT3; //off
        P7OUT = P7OUT & ~BIT3; //off
        P9OUT = P9OUT & ~BIT5; //off
        P6OUT = P6OUT & ~BIT3; //off
        ////////////////////////////////
        P7OUT = P7OUT | BIT0; //on top middle
        P5OUT = P5OUT | BIT3; //on top right
        P9OUT = P9OUT | BIT5; //on middle segment
        P8OUT = P8OUT | BIT6; //on lower left
        P8OUT = P8OUT | BIT7; //on lower middle
    }
    if (result < 4000 && result > 3000)
    {
        ////////////////////////////////
        P7OUT = P7OUT & ~BIT0; //off
        P9OUT = P9OUT & ~BIT1; //off
        P8OUT = P8OUT & ~BIT7; //off
        P8OUT = P8OUT & ~BIT6; //off
        P8OUT = P8OUT & ~BIT3; //off
        P5OUT = P5OUT & ~BIT3; //off
        P7OUT = P7OUT & ~BIT3; //off
        P9OUT = P9OUT & ~BIT5; //off
        P6OUT = P6OUT & ~BIT3; //off
        ////////////////////////////////
        P7OUT = P7OUT | BIT0; //on top middle
```

```
            P5OUT = P5OUT | BIT3; //on top right
            P9OUT = P9OUT | BIT5; //on middle segment
            P9OUT = P9OUT | BIT1; //on lower right
            P8OUT = P8OUT | BIT7; //on lower middle
        }
        if (result < 5000 && result > 4000)
        {
            ////////////////////////////////////
            P7OUT = P7OUT & ~BIT0; //off
            P9OUT = P9OUT & ~BIT1; //off
            P8OUT = P8OUT & ~BIT7; //off
            P8OUT = P8OUT & ~BIT6; //off
            P8OUT = P8OUT & ~BIT3; //off
            P5OUT = P5OUT & ~BIT3; //off
            P7OUT = P7OUT & ~BIT3; //off
            P9OUT = P9OUT & ~BIT5; //off
            P6OUT = P6OUT & ~BIT3; //off
            ////////////////////////////////////

            P9OUT = P9OUT | BIT1; //on lower right
            P8OUT = P8OUT | BIT3; //on dot
            P5OUT = P5OUT | BIT3; //on top right
            P7OUT = P7OUT | BIT3; //on top left
            P9OUT = P9OUT | BIT5; //on middle segment

        }
        if (result < 6000 && result > 5000)
        {
            ////////////////////////////////////
            P7OUT = P7OUT & ~BIT0; //off
            P9OUT = P9OUT & ~BIT1; //off
            P8OUT = P8OUT & ~BIT7; //off
            P8OUT = P8OUT & ~BIT6; //off
            P8OUT = P8OUT & ~BIT3; //off
            P5OUT = P5OUT & ~BIT3; //off
            P7OUT = P7OUT & ~BIT3; //off
            P9OUT = P9OUT & ~BIT5; //off
            P6OUT = P6OUT & ~BIT3; //off
            ////////////////////////////////////
            P7OUT = P7OUT | BIT0; //on top middle
            P9OUT = P9OUT | BIT1; //on lower right
            P8OUT = P8OUT | BIT7; //on lower middle
            P8OUT = P8OUT | BIT3; //on dot
            P7OUT = P7OUT | BIT3; //on top left
            P9OUT = P9OUT | BIT5; //on middle segment
            P6OUT = P6OUT | BIT3; //on top middle
        }
        if (result < 7000 && result > 6000)
        {
            ////////////////////////////////////
            P7OUT = P7OUT & ~BIT0; //off
            P9OUT = P9OUT & ~BIT1; //off
            P8OUT = P8OUT & ~BIT7; //off
            P8OUT = P8OUT & ~BIT6; //off
            P8OUT = P8OUT & ~BIT3; //off
            P5OUT = P5OUT & ~BIT3; //off
            P7OUT = P7OUT & ~BIT3; //off
            P9OUT = P9OUT & ~BIT5; //off
            P6OUT = P6OUT & ~BIT3; //off
            ////////////////////////////////////

            P7OUT = P7OUT | BIT0; //on top middle
            P9OUT = P9OUT | BIT1; //on lower right
            P8OUT = P8OUT | BIT7; //on lower middle
            P8OUT = P8OUT | BIT6; //on lower left
            P8OUT = P8OUT | BIT3; //on dot
            P9OUT = P9OUT | BIT5; //on middle segment
            P7OUT = P7OUT | BIT3; //on top left
            P6OUT = P6OUT | BIT3; //on top middle

        }
        if (result < 8000 && result > 7000)
        {
            ////////////////////////////////////
            P7OUT = P7OUT & ~BIT0; //off
            P9OUT = P9OUT & ~BIT1; //off
            P8OUT = P8OUT & ~BIT7; //off
            P8OUT = P8OUT & ~BIT6; //off
            P8OUT = P8OUT & ~BIT3; //off
            P5OUT = P5OUT & ~BIT3; //off
            P7OUT = P7OUT & ~BIT3; //off
            P9OUT = P9OUT & ~BIT5; //off
            P6OUT = P6OUT & ~BIT3; //off
            ////////////////////////////////////
            P7OUT = P7OUT | BIT0; //on top middle
            P9OUT = P9OUT | BIT1; //on lower right
            P8OUT = P8OUT | BIT3; //on dot
            P5OUT = P5OUT | BIT3; //on top right

        }
        if (result < 9000 && result > 8000)
        {
            ////////////////////////////////////
            P7OUT = P7OUT & ~BIT0; //off
            P9OUT = P9OUT & ~BIT1; //off
            P8OUT = P8OUT & ~BIT7; //off
            P8OUT = P8OUT & ~BIT6; //off
            P8OUT = P8OUT & ~BIT3; //off
            P5OUT = P5OUT & ~BIT3; //off
            P7OUT = P7OUT & ~BIT3; //off
            P9OUT = P9OUT & ~BIT5; //off
            P6OUT = P6OUT & ~BIT3; //off
            ////////////////////////////////////
            P7OUT = P7OUT | BIT0; //on top middle
            P9OUT = P9OUT | BIT1; //on lower right
            P8OUT = P8OUT | BIT7; //on lower middle
            P8OUT = P8OUT | BIT6; //on lower left
            P8OUT = P8OUT | BIT3; //on dot
            P5OUT = P5OUT | BIT3; //on top right
            P7OUT = P7OUT | BIT3; //on top left
            P9OUT = P9OUT | BIT5; //on middle segment
            P6OUT = P6OUT | BIT3; //on top middle

        }
        if (result < 12000 && result > 9000)
        {
            ////////////////////////////////////
            P7OUT = P7OUT & ~BIT0; //off
            P9OUT = P9OUT & ~BIT1; //off
            P8OUT = P8OUT & ~BIT7; //off
            P8OUT = P8OUT & ~BIT6; //off
            P8OUT = P8OUT & ~BIT3; //off
            P5OUT = P5OUT & ~BIT3; //off
            P7OUT = P7OUT & ~BIT3; //off
            P9OUT = P9OUT & ~BIT5; //off
            P6OUT = P6OUT & ~BIT3; //off
            ////////////////////////////////////
            P7OUT = P7OUT | BIT0; //on top middle
            P9OUT = P9OUT | BIT1; //on lower right
            P8OUT = P8OUT | BIT3; //on dot
            P5OUT = P5OUT | BIT3; //on top right
            P7OUT = P7OUT | BIT3; //on top left
            P9OUT = P9OUT | BIT5; //on middle segment
            P6OUT = P6OUT | BIT3; //on top middle
        }
        ////////////////////////////////////////////
    }

    }       //end of while
}       //end main
////////////////////////////////////////////////////////////////END MAIN//

void playGame()
{
```

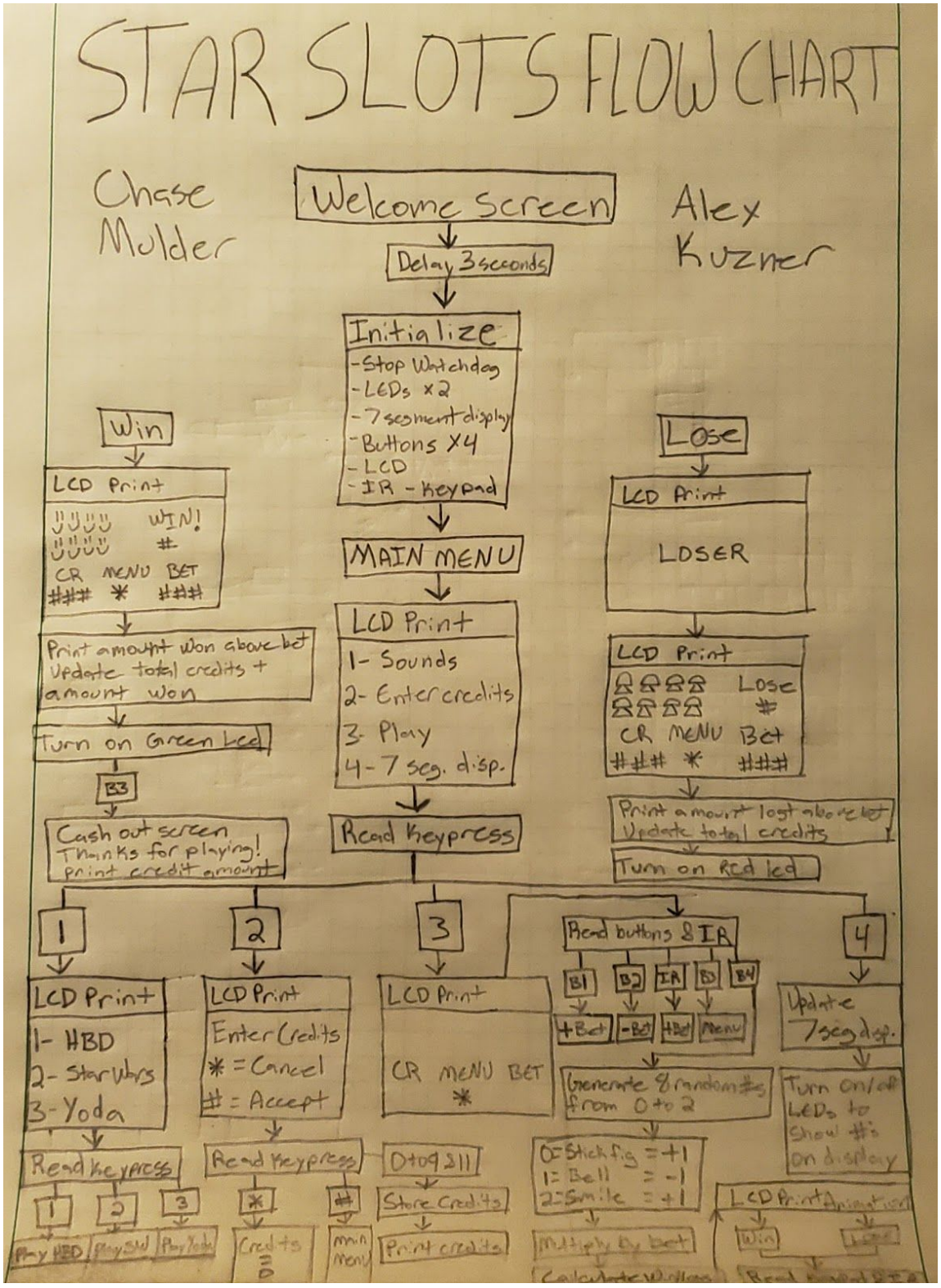
```
    char stringGame[16] = "                ";
    LCD_CommandWrite(0x90);
    strcpy(stringGame, "CR     Menu  BET");
    printString(stringGame);
    LCD_CommandWrite(0xD0);
    strcpy(stringGame, "        *       ");
    printString(stringGame);

    while (1)
    {

        print_Array(); //Credits in lower left
        lcdSetInt(credits_array[0], 13, 3);
        lcdSetInt(credits_array[1], 14, 3);
        lcdSetInt(credits_array[2], 15, 3); //Bet in lower right
        SwitchStatus_Launchpad_Button1();
        SwitchStatus_Launchpad_Button2();
        SwitchStatus_Launchpad_Button3();
        SwitchStatus_Launchpad_Button4();

        ////////////////////////////////////////////////////////////

        if (P4IN & BIT3)
        {
            printf("working coin\n");

            newCred = newCred + 1;
            printf("working\n");
            P1->DIR |= BIT7;
            P1->OUT &= ~BIT7;
            P1->DIR &= ~BIT4;

            for (i = 0; i < 2; i++)
            {
                Sound_Play(8 * notes3[i], 100 * interval3[i]);
                pause(6);
            }

            button_status = 0;
            P4IN &= ~BIT3;
            print_Array();
            playGame();

        }

        ////////////////////////////////////////////////////////////
        if (SwitchStatus_Launchpad_Button1() == PRESSED)
        { //Button debouncing
            SysTick_delay_ms(3);
            if (SwitchStatus_Launchpad_Button1() == PRESSED)
            {
                button_status = 1;
            }
        }

        if (button_status == 1)
        {
            printf("work\n");
            if (credits_array[2] < arraykeys[0] * 100
                  & credits_array[2] < arraykeys[1] * 10
                  & credits_array[2] < 5)
            {
                credits_array[2] = credits_array[2] + 1;
                printf("working\n");
                button_status = 0;
                SysTick_delay_ms(10);
                playGame();
            }

            button_status = 0;

        }
        ////////////////////////////////////////////////////////////
        if (SwitchStatus_Launchpad_Button2() == PRESSED)
        { //Button debouncing
            SysTick_delay_ms(3);
            if (SwitchStatus_Launchpad_Button2() == PRESSED)
            {
                button_status = 1;
            }
        }

        if (button_status == 1)
        {
            printf("work\n");
            if (credits_array[2] > 0)
            {
                credits_array[2] = credits_array[2] - 1;
                printf("working\n");
                button_status = 0;
                SysTick_delay_ms(10);
                playGame();
            }

            button_status = 0;
        }

        ////////////////////////////////////////////////////////////
        if (SwitchStatus_Launchpad_Button3() == PRESSED)
        {
            main_Menu();
        }
        ////////////////////////////////////////////////////////////
        if (SwitchStatus_Launchpad_Button4() == PRESSED)
        { //Button debouncing
            SysTick_delay_ms(3);
            if (SwitchStatus_Launchpad_Button4() == PRESSED)
            {
                button_status = 1;
            }
        }

        if (button_status == 1)
        {
            button_status = 0;
            Spin();
        }
        ////////////////////////////////////////////////////////////
    } //end while
} //end playGame

void Sound_Play(unsigned freq_in_hz, unsigned duration_ms)
{
    uint32_t i = 0;
    float time_period_ms = (1.0 / freq_in_hz) * 1000000.0;
    for (i = 0; i < duration_ms; i++)
    {
        P1->OUT |= BIT7;
        SysTick_delay_us(time_period_ms);
        P1->OUT &= ~BIT7;
        SysTick_delay_us(time_period_ms);
    }
}
void pause(unsigned short i)
{
    unsigned short j;
    for (j = 0; j < i; j++)
        SysTick_delay_ms(10);
}

void initialize()
{
    //Stop watchdog timer
```

```
    WDT_A_hold(WDT_A_BASE);

    led_source();

    seg_source();

    button_source();

    lcd_source();

    ir_source();

    keypad_source();

}

void storeArray()
{ //Bit shift left when more than 3 inputs have been inputted

    for (i = 0; i < 3;)
    {
        keypress = Read_Keypad();

        if (keypress == 1)
        {
            arraykeys[i] = keypress;
            i++;

            keypress = 13;
        }
        else if (keypress == 2)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 3)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 4)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 5)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 6)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 7)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 8)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 9)
        {
            arraykeys[i] = keypress;
            i++;
            keypress = 13;
        }
        else if (keypress == 10)
        {

            arraykeys[0] = 0;
            arraykeys[1] = 0;
            arraykeys[2] = 0;
            i = 0;
            keypress = 13;
        }
        else if (keypress == 11)
        {
            arraykeys[i] = 0;
            i++;
        }
        else if (keypress == 12)
        {
            main_Menu();
            break;
        }

    }
    i = 0;

    for (i = 0; i < 3; i++)
    {
        if (i == 0)
        {
            newCred = arraykeys[0];
        }
        if (i == 1)
        {
            newCred = arraykeys[0] * 10;
            newCred += arraykeys[1];
        }
        if (i == 2)
        {
            newCred = arraykeys[0] * 100;
            newCred += arraykeys[1] * 10;
            newCred += arraykeys[2];
        }
    }

    print_Array();

    storeArray();
}

void print_Array()
{

    lcdSetInt(newCred, 0, 3);

}
```

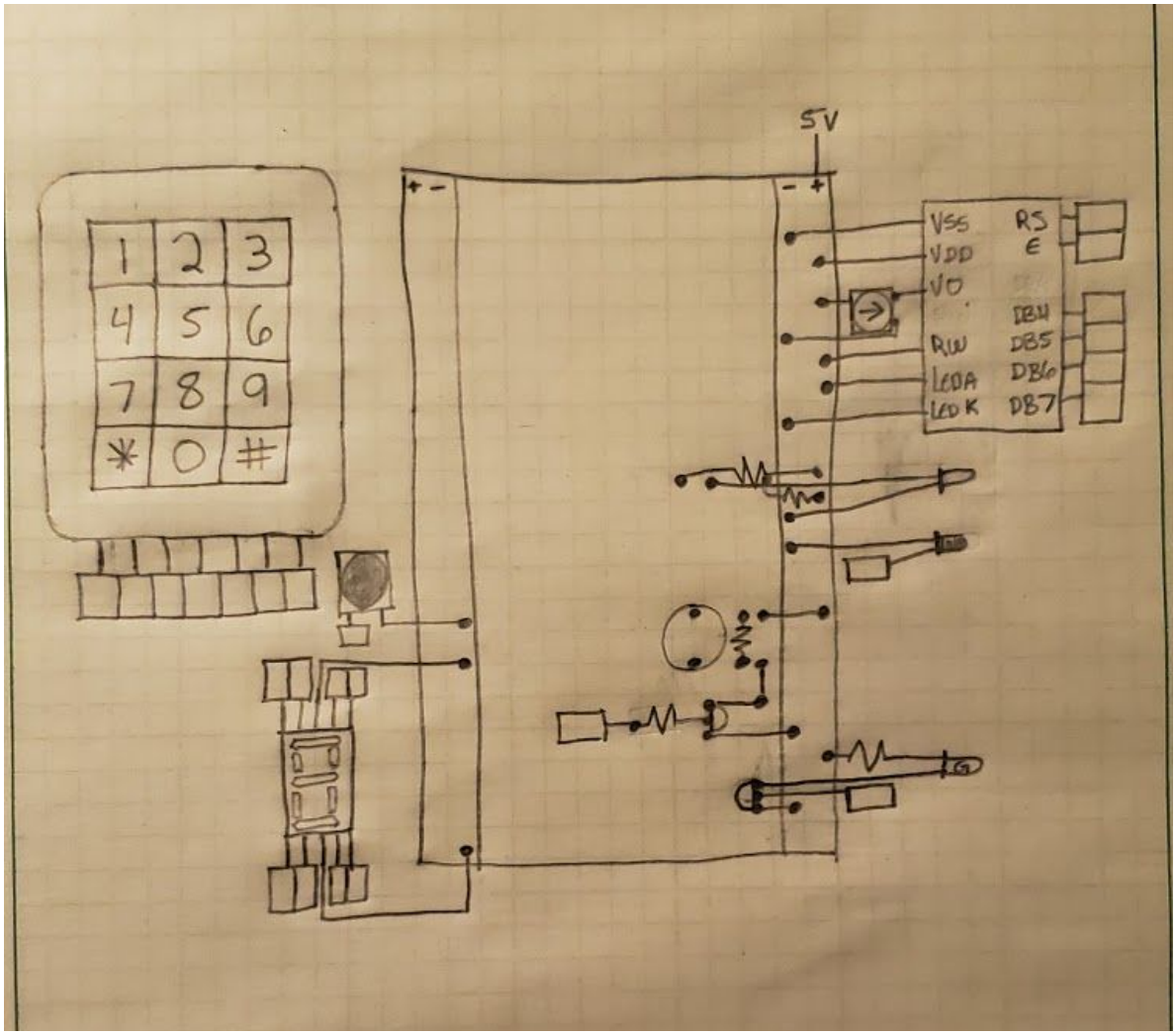
Conclusion:

The final project included many different challenges and lessons. The project incorporated all of the knowledge gained during the semester. Determining the material and the kind of box to make for the slot machine was a difficult task until the perfect fitting boxes were seen. The idea of the slot machine was a very interesting way to incorporate our newly found knowledge and skills to build something that is used everyday in the real world even if it's just a gambling device. The hardest and most difficult part dealt with the coding. The code was an incorporation of all the labs done during the semester and put together to make a slot machine work. The circuit wasn't anything new so it was easier mentally but the task took a long time to make correctly, organized and able to fit in the slot machine. In the completion of the project it was more than just a grade or just points. It is a completion of a project started and finished that was created from scratch. The star wars theme and the completion of the project gave a warm happy feeling of success that acknowledged the hard work and new skills learned all semester.

Appendix A


STAR SLOTS FLOW CHART
Chase Mulder
Alex Kuzner
Welcome screen
Delay 3 seconds
Initialize
-Stop Watchdog
-LEDs x2
-7 segment display
-Buttons X4
-LCD
-IR - keypad
MAIN MENU
LCD Print
1- Sounds
2- Enter credits
3- Play
4- 7 seg. disp.
Read keypress
Win
LCD Print
WIN!
CR MENU BET
Print amount won above bet
Update total credits + amount won
Turn on Green Led
B3
Cash out screen
Thanks for playing!
Print credit amount
Lose
LCD Print
LOSER
LCD Print
Lose
CR MENU Bet
Print amount lost above bet
Update total credits
Turn on Red led
1
LCD Print
1- HBD
2- Star Wars
3- Yoda
Read keypress
2
LCD Print
Enter Credits
* = Cancel
# = Accept
Read keypress
3
LCD Print
CR MENU BET
*
0-9 and #
Store Credits
Print credits
Read buttons & IR
B1
B2
IR
B3
B4
+Bet
-Bet
+Bet
Menu
Generate 3 random #s from 0 to 2
0=Stick tongue = +1
1=Bell = -1
2=Smile = +1
Multiply by bet
4
Update 7seg disp.
Turn on/off LEDs to show #'s on display
LCD Print Animation
Win
Lose

Appendix B



7 Seg Disp: 3&8GND, P9.5, P7.0, P7.3, P6.3, P5.3, P8.3, P9.1, P8.7, P8.6
Keypad Columns: P5.0, P5.1, P5.2
Keypad Rows: P2.4, P2.5, P2.6, P2.7
Button 1: P6.0
Button 2: P6.1
Button 3: P3.2
Button 4: P3.3
LCD DB: 4P4.4, 5P4.5, 6P4.6, 7P4.7
LCD RS: P4.0
LCD E: P4.1
Green LED: P3.6
Red LED: P6.9
Piezo Buzzer: P1.7

Appendix Sound

```
            // Drive buzzer with P1.7
            P1->DIR |= BIT7;
            P1->OUT &= ~BIT7;
            P1->DIR &= ~BIT4;
            for (i = 0; i < 25; i++)
            {
               Sound_Play(8 * notes[i], 100 * interval[i]);
               pause(6);
            }
            main_Menu();
            pause(100);
         }
//////SONGS////////////////////////////////////////////////////////////////////////
// Happy birthday notes
/*                Hap py  Birth Day  to  you,  Hap py  birth day  to
 C4   C4   D4   C4   F4   E4   C4   C4   D4   C4   G4 */
unsigned int notes[] = { 262, 262, 294, 262, 349, 330, 262, 262, 294, 262, 392,

/*                you, Hap py  Birth Day  dear  xxxx      Hap  py   birth
 F4   C4   C4   C5   A4   F4   E4   D4   B4b  B4b  A4 */
349,
                262, 262, 523, 440, 349, 330, 294, 466, 466, 440,

                /*                day  to  you
                 F4   G4   F4   */
                349,
                392, 349 };

unsigned short interval[] = { 4, 4, 8, 8, 8, 10, 4, 4, 8, 8, 8, 10, 4, 4, 8, 8, 8, 8, 8, 4, 4, 8, 8, 8, 12 };
#define c 261
#define d 294
#define e 329
#define f 349
#define g 391
#define gS 415
#define a 440
#define aS 455
#define b 466
#define cH 523
#define cSH 554
#define dH 587
#define dSH 622
#define eH 659
#define fH 698
#define fSH 740
#define gH 784
#define gSH 830
#define aH 880
// STAR WARS
unsigned int notes2[] = { a, a, a, f, cH, a, f, cH, a };
unsigned short interval2[] = { 4, 4, 8, 8, 8, 10, 4, 4, };
//Yoda noise
unsigned int notes3[] = { a, f };
unsigned short interval3[] = { 20, 20 };
```

Appendix LEDs

```
#ifndef LED_SOURCE_H_
#define LED_SOURCE_H_

void led_source();
void led_source(){
   //Initializing green LED
   uint8_t greenLED = BIT6;
   P3SEL1 &= ~greenLED;
   P3SEL0 &= ~greenLED; //Green
   P3DIR |= greenLED;
   //Initializing red LED
   uint8_t redLED = BIT9;
   P6SEL1 &= ~redLED;
   P6SEL0 &= ~redLED; //Red
   P6DIR |= redLED;
   //LEDS off
   P2OUT &= ~redLED;
   P2OUT &= ~greenLED;
   }

void TurnOn_Green_LED()
{
   P3OUT |= P3OUT | BIT6;
}
void TurnOff_Green_LED()
{
   P6OUT = P6OUT & ~BIT6;
}
void TurnOn_Red_LED()
{
   P6OUT |= P6OUT | BIT7;
}
void TurnOff_Red_LED()
{
   P6OUT = P6OUT & ~BIT7;
}


#endif /* LED_SOURCE_H_ */
```

Appendix Buttons

```
#ifndef BUTTON_SOURCE_H_
#define BUTTON_SOURCE_H_

void button_source();
void button_source(){
   //Initializing Button 1
   P6DIR &= ~BIT0;
   P6REN |= BIT0;
   P6OUT |= BIT0;

   //Initializing Button 2
   P6DIR &= ~BIT1;
   P6REN |= BIT1;
   P6OUT |= BIT1;
   //Initializing Button 3
   P3DIR &= ~BIT2;
   P3REN |= BIT2;
   P3OUT |= BIT2;
   //Initializing Button 4
   P3DIR &= ~BIT3;
   P3REN |= BIT3;
   P3OUT |= BIT3;

}



char SwitchStatus_Launchpad_Button1()
{
   return (P6IN & BIT0 );
}
char SwitchStatus_Launchpad_Button2()
{
   return (P6IN & BIT1 );
}
char SwitchStatus_Launchpad_Button3()
{
   return (P3IN & BIT2 );
}
char SwitchStatus_Launchpad_Button4()
{
   return (P3IN & BIT3 );
}


#endif /* BUTTON_SOURCE_H_ */
```

Appendix LCD

```
#ifndef LCD_SOURCE_H_
#define LCD_SOURCE_H_
void lcd_source();
void lcd_source(){
    //LCD
    uint8_t LEDrs = BIT0;
    P4SEL1 &= ~LEDrs;
    P4SEL0 &= ~LEDrs; //RS
    P4DIR |= LEDrs;
    P4OUT &= ~LEDrs;
    uint8_t LEDe = BIT1;
    P4SEL1 &= ~LEDe;
    P4SEL0 &= ~LEDe; //E
    P4DIR |= LEDe;
    P4OUT &= ~LEDe;
    uint8_t LEDdb4 = BIT4;
    P4SEL1 &= ~LEDdb4;
    P4SEL0 &= ~LEDdb4; //DB4
    P4DIR |= LEDdb4;
    P4OUT &= ~LEDdb4;
    uint8_t LEDdb5 = BIT5;
    P4SEL1 &= ~LEDdb5;
    P4SEL0 &= ~LEDdb5; //DB5
    P4DIR |= LEDdb5;
    P4OUT &= ~LEDdb5;
    uint8_t LEDdb6 = BIT6;
    P4SEL1 &= ~LEDdb6;
    P4SEL0 &= ~LEDdb6; //DB6
    P4DIR |= LEDdb6;
    P4OUT &= ~LEDdb6;
    uint8_t LEDdb7 = BIT7;
    P4SEL1 &= ~LEDdb7;
    P4SEL0 &= ~LEDdb7; //DB7
    P4DIR |= LEDdb7;
    P4OUT &= ~LEDdb7;
    LCD_PushByte(0x08);
    SysTick_delay_us(100000);
    LCD_PushByte(0x30);
    SysTick_delay_us(100000);
    LCD_PushByte(0x30);
    SysTick_delay_us(100000);
    LCD_PushByte(0x30);
    SysTick_delay_us(100000);
    LCD_PushByte(0x02);
    SysTick_delay_us(100000);
    LCD_PushByte(0x06);
    SysTick_delay_us(100000);
    LCD_PushByte(0x01);
    SysTick_delay_us(100000);
    LCD_PushByte(0x0F);
    SysTick_delay_us(100000);
    }
```

```
void LCD_PulseEnable(void)
{
   P4OUT &= ~BIT1;
   SysTick_delay_us(10);
   P4OUT |= BIT1;
   SysTick_delay_us(10);
   P4OUT &= ~BIT1;
}
void LCD_PushNibble(uint8_t nibble)
{
   P4OUT &= ~(0xF0); // ASSUMPTION: D7-D4 are on P4.7=4.4
   P4OUT |= (nibble & 0x0F) << 4; // output nibble value on port pins P4.7-4.4
   LCD_PulseEnable();
}
void LCD_PushByte(uint8_t byte)
{
   uint8_t upper;
   uint8_t lower;
   upper = (byte & 0xF0) >> 4; // mask upper 4 and shift 4 to the right
   lower = (byte & 0x0F); // mask lower 4 and keep
   LCD_PushNibble(upper); // push upper nibble first
   LCD_PushNibble(lower); // «+. then lower nibble
   SysTick_delay_us(100);
}

void LCD_CommandWrite(uint8_t command)
{
   P4OUT &= ~BIT0;
   LCD_PushByte(command);
   //LCD_PulseEnable();
}
void LCD_DataWrite(uint8_t data)
{
   P4OUT |= BIT0;
   LCD_PushByte(data);
   //LCD_PulseEnable();
}


#endif /* LCD_SOURCE_H_ */
```

Appendix 7 Segment Display

```
#ifndef SEG_SOURCE_H_
#define SEG_SOURCE_H_

void seg_source();
void seg_source(){

    //7 segment display leds

    //
    P9SEL1 &= ~BIT5;
    P9SEL0 &= ~BIT5;
    P9DIR |= BIT5;
    P9OUT &= ~BIT5;
    //
    P7SEL1 &= ~BIT0;
    P7SEL0 &= ~BIT0;
    P7DIR |= BIT0;
    P7OUT &= ~BIT0;
    //
    P7SEL1 &= ~BIT3;
    P7SEL0 &= ~BIT3;
    P7DIR |= BIT3;
    P7OUT &= ~BIT3;
    //
    P6SEL1 &= ~BIT3;
    P6SEL0 &= ~BIT3;
    P6DIR |= BIT3;
    P6OUT &= ~BIT3;
    //
    P5SEL1 &= ~BIT3;
    P5SEL0 &= ~BIT3;
    P5DIR |= BIT3;
    P5OUT &= ~BIT3;
    //
    P8SEL1 &= ~BIT3;
    P8SEL0 &= ~BIT3;
    P8DIR |= BIT3;
    P8OUT &= ~BIT3;
    //
    P9SEL1 &= ~BIT1;
    P9SEL0 &= ~BIT1;
    P9DIR |= BIT1;
    P9OUT &= ~BIT1;
    //
    P8SEL1 &= ~BIT7;
    P8SEL0 &= ~BIT7;
    P8DIR |= BIT7;
    P8OUT &= ~BIT7;
    //
    P8SEL1 &= ~BIT6;
    P8SEL0 &= ~BIT6;
    P8DIR |= BIT6;
    P8OUT &= ~BIT6;
}
```

```
void ADC14_pinInit( Void)
{
   P5->SEL1 |= BIT5;
   P5->SEL0 |= BIT5;
   P5->DIR &= ~ BIT5;
}
void ADC14_preiphInit(void)
{
   ADC14->CTL0 &= ~ ADC14_CTL0_ENC;
   ADC14->CTL0 |= 0x04200210;
   ADC14->CTL1 = 0x00000030;
   ADC14->MCTL[0] = 0x00000000;
   ADC14->CTL0 |= ADC14_CTL0_ENC;
}



#endif /* SEG_SOURCE_H_ */
```

Appendix Keypad

```
#ifndef KEYPAD_SOURCE_H_
#define KEYPAD_SOURCE_H_

void keypad_source();
void keypad_source(){
   //initialize rows
   P2SEL0 &= ~(BIT4 | BIT5 | BIT6 | BIT7 );
   P2SEL1 &= ~(BIT4 | BIT5 | BIT6 | BIT7 );
   P2DIR &= ~(BIT4 | BIT5 | BIT6 | BIT7 );
   P2REN |= (BIT4 | BIT5 | BIT6 | BIT7 );
   P2OUT |= (BIT4 | BIT5 | BIT6 | BIT7 );
   }
void printString(char stringType[])
{
   int i;
   for (i = 0; i < 16; i++)
   {
      char letter = stringType[i];
      LCD_DataWrite(letter);
   }
}
int Read_Keypad()
{
   uint8_t row, col;
   for (col = 0; col < 3; col++)
   {
      P5->DIR &= ~(BIT0 | BIT1 | BIT2 ); //Initialize columns port 5 bits 0,1,2
      P5->DIR |= (1 << (col));
      P5->OUT &= ~(1 << (col));
      SysTick_delay_ms(10);
      row = P2->IN & 0xF0;
      while (!(P2->IN & BIT4 ) | !(P2->IN & BIT5 ) | !(P2->IN & BIT6 )
            | !(P2->IN & BIT7 ))
         ; //Initialize rows port 6 bits 0, 1, 4, 5
      if (row != 0xF0)
         break;
   }
   P5->DIR &= ~(BIT0 | BIT1 | BIT2 );
   if (col == 3)
      return 0;
   if (row == 0b11100000)
      return col + 1;
   if (row == 0b11010000)
      return 3 + col + 1;
   if (row == 0b10110000)
      return 6 + col + 1;
   if (row == 0b01110000)
      return 9 + col + 1;

   return -1;

}
#endif /* KEYPAD_SOURCE_H_ */
```

Appendix IR

```
#ifndef IR_SOURCE_H_
#define IR_SOURCE_H_
void ir_source();
void ir_source(){
    //IR sensor
    P4->SEL0 &= ~BIT3;
    P4->SEL1 &= ~BIT3;
    P4->DIR &= ~BIT3;
    P4DIR &= ~BIT3;
    P4REN |= BIT3;
    P4OUT |= BIT3;
    }


#endif /* IR_SOURCE_H_ */
```