Lab Report 1

Jack Christopher Bergin

ECE 2049 - C21’

Professor Doroz Yarkin

ECE2049 C-2021
Lab 1 Sign-off Sheet

***Report due*: Tuesday 3/17/21**

Student 1: Jack Bergin **ECE mailbox:** _____

Student 2: _____ **ECE mailbox:** _____

Task Max points	***Max points***	***TA's assessment***
SIMON on LCD	10	
Game Restart ('*' pressed together)	10	
Display random sequence on LCD (length of at least 10)	15	
Display player's button press on LCD, spatially aligned with button	15	
Play complete game (i.e. play sequence and check sequence correctly) BONUS: Increase speed as you go	15 (+5)	
BONUS: Have buzzer play different pitch for each LED	(+5)	
Proper player humiliation on error & reset to welcome screen	5	
Report	30	
Total points (with bonuses)	100 ***110***	

TA's signature: _____ **Date:** _____

***** Both Students MUST be present at Sign-off for any and all parts!!***

Introduction:

In this lab the objective was to create a game called “Simon” that went off the basic principles of ‘Simon Says’. An array of numbers would be displayed on the LCD (the amount of numbers beginning at 1 and then iterating to a max length of 10) and then erased. This is where the user would have to input their guess of the numbers displayed on the LCD. With each button press on the number pad a buzzer would be engaged. Depending on if the person gets the guess right, the game will either keep going, redirect to the losing screen before returning back to the home screen when ‘#’ is pressed, or play again with the ‘*’ key.

Discussion and Results:

When initially looking at the problem I could see how it could be done with a state machine but the way I thought to parse through the simon value array was through nested for loops. Once I got the game logic done I was able to structure code around the for loops to complete the other actions important to the lab requirements. The functions I defined for this program were used multiple times in the looped sequence of the program so to save space and consolidate they were created. One of the biggest struggles within this lab was creating values with the same data types to have the ability to print out onto the LCD while also being able to be compared to one another. For this function, the simon values were specified to a pointer in order to convert the random assigned simon array values to values comparable to the keypad output.

Next, the newly defined simon array was sent to a method called simonValues where it would properly be defined with 10 values spanning from 1-4 with simonArray with length 10. The reason why there needs a maximum length for the games simon number sequence so that the

game can be won and because with the usage of arrays, a specific length has to be set in order for the program to work.

Once the values are established, we enter the nested for loop portion of the code that makes up the game logic. Each run of the interior for loop will print out the simon values corresponding to the outer loop, listen for an answer back with the getMyKey() function, and from there determine if the guess was correct, incorrect, or if the game limit was hit. If the guess was correct, the loops will iterate and the array will print an additional character. If the guess is wrong, then the code will exit to a new screen specifying that you lost and then gives the options to press '*' for a rematch and '#' to get back to the 'Simon' screen. If the loop hits the array length limit then the code will exit to another new screen specifying victory in the game, giving the option for exiting to the 'Simon' screen.

Code Breakdown and annotation:

NOTE: All of the text in bold is my annotation for the code, what each function does what it does, why it is needed, and how it ties into the overall goal of the lab.

```
//Libraries
#include "peripherals.h"
#include "stdbool.h"
#include <msp430.h>

// These statements are the Function Prototypes
//This gives a delay in the program
void swDelay(char numLoops);

//This organizes the getKey() function and makes the time delay for the key value to be
//collected infinite so that the key is pressed once with only one value rather than getting
//pressed once and reading multiple values.
unsigned char getMyKey(void);

//This function prints the “Ready? 3… 2… 1… Go!” countdown by painting to the LCD
//screen, delaying, erasing, painting, refreshing, delaying, etc with the process being
//repeated 5 times.
void countDown();
```

```
//This defines the simon values for the simonArray and updates how many values will be
//utilized based on the logic in the nested for loop hosting the main game logic.
unsigned char simonValues(int numLoops, unsigned char *simonArray);

//This is also used in the game logic loops as a way to print out each simonArray value and
//then refresh accordingly. To print out properly, it uses a switch case statement to go
//through each value stored in the simonArray and then print out the corresponding value
//on the LCD. This is due to the formatting that the LCD likes for data types.
void simonDisplay(int numLoops, unsigned char *simonArray);

//This command defines all loop variables
volatile unsigned int i,j,k,m,n;

void main(void)
{
   //Pre-program setup
   WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
   __disable_interrupt();
   configDisplay();
   configKeypad();
```

```
//Paints the intro screen for 'Simon'
Graphics_clearDisplay(&g_sContext);
Graphics_drawStringCentered(&g_sContext, "Simon", AUTO_STRING_LENGTH, 64, 50,
TRANSPARENT_TEXT);
Graphics_drawStringCentered(&g_sContext, "Press '*' to play!", AUTO_STRING_LENGTH,
64, 60, TRANSPARENT_TEXT);
Graphics_flushBuffer(&g_sContext);

//Begins this infinite while loop to constantly listen for the keys to initiate the game and
then to play the game all of the way through.
while (1) {
    unsigned char currKey = getMyKey();

    //When the game is over and the player wants the LCD refreshed, they can press the
    //'#' character on the keyboard. This will clear and then write intro screen painted
    //before entering this while loop.
    if (currKey == '#') {
        //Paints the intro screen after the game
        Graphics_clearDisplay(&g_sContext); // Clear the display
        Graphics_drawStringCentered(&g_sContext, "Simon", AUTO_STRING_LENGTH, 64,
        50, TRANSPARENT_TEXT);
        Graphics_drawStringCentered(&g_sContext, "Press '*' to play!",
```

```
    AUTO_STRING_LENGTH, 64, 60, TRANSPARENT_TEXT);
    Graphics_flushBuffer(&g_sContext);
}

if (currKey == '*')  {
    //Paints the screen for the count down of the game
    countDown();

    //Establishes the pointer values for the simonArray so that formatting is uniform
    //and conversion arrays are not needed. We don't want to use conversion arrays
    //because they can throw unneeded complexity into the loops for this program.
    unsigned char *simonArray;
    for(i = 0; i < 10; i++) {
        free(simonArray[i]);
    }
    int simonLength = 10;
    simonArray = (int *)malloc(simonLength * sizeof(int));

    //This command sends the simonArray to this method for its values to get created
    //and set.
    simonValues(simonLength, simonArray);
```

```
//The following loops are for the 'Simon' game logic portion of the code. In this there
//is a nested for loop to run both the graphics and game rules of the program.
for(j = 0; j < 10; j++){
    //This command will display each character on to the LCD according to the
    //simonDisplay() function.
    simonDisplay(j, simonArray);
    for(k = 0; k < j; k++){
      //This is for my input key from keypad for the simon game
      unsigned char myKey = getMyKey();


      //Logic for comparing the displayed simon value to the keypad input
      if (myKey == simonArray[k]){
        switch(myKey){
        //Each case will print out the key value entered as well as the engaging the
        //buzzer to the corresponding key. The statements in each case are very
        //similar so a function could be implemented here to clean up the code
        //further.
        case '4':
          //Paints to LCD
          Graphics_clearDisplay(&g_sContext);
          Graphics_drawStringCentered(&g_sContext, "4",
          AUTO_STRING_LENGTH, 88, 50, TRANSPARENT_TEXT);
```

```
    Graphics_flushBuffer(&g_sContext);

    //Turns on and off the buzzer for one second
    BuzzerOn(4);
    swDelay(1);
    BuzzerOff();

    break;

case '3':
    //Paints to LCD
    Graphics_clearDisplay(&g_sContext);
    Graphics_drawStringCentered(&g_sContext, "3",
    AUTO_STRING_LENGTH, 66, 50, TRANSPARENT_TEXT);
    Graphics_flushBuffer(&g_sContext);

    //Turns on and off the buzzer for one second
    BuzzerOn(3);
    swDelay(1);
    BuzzerOff();

    Break;
```

```
case '2':
    //Paints to LCD
    Graphics_clearDisplay(&g_sContext);
    Graphics_drawStringCentered(&g_sContext, "2",
    AUTO_STRING_LENGTH, 44, 50, TRANSPARENT_TEXT);
    Graphics_flushBuffer(&g_sContext);

    //Turns on and off the buzzer for one second
    BuzzerOn(2);
    swDelay(1);
    BuzzerOff();

    break;

case '1':
    //Paints to LCD
    Graphics_clearDisplay(&g_sContext);
    Graphics_drawStringCentered(&g_sContext, "1",
    AUTO_STRING_LENGTH, 22, 50, TRANSPARENT_TEXT);
    Graphics_flushBuffer(&g_sContext);
```

```
        //Turns on and off the buzzer for one second
        BuzzerOn(1);
        swDelay(1);
        BuzzerOff();


        break;
    }
    swDelay(1);


}
else if (myKey == 0)  {
    //If the key is 0 then the function will just loop back through
}
//Logic for if the game is lost and the wrong button was pressed.
else if((myKey != simonArray[k]) || (myKey != 0)){
    //Force exits the loops
    j = 11;
    k = 11;


    //Delays shortly
    swDelay(1);
```

```
        //Paints to the LCD screen
        Graphics_clearDisplay(&g_sContext);
        Graphics_drawStringCentered(&g_sContext, "Game Over!",
        AUTO_STRING_LENGTH, 64, 50, TRANSPARENT_TEXT);
        Graphics_drawStringCentered(&g_sContext, "You Lost!",
        AUTO_STRING_LENGTH, 64, 60, TRANSPARENT_TEXT);
        Graphics_drawStringCentered(&g_sContext, "Press '#' for home",
        AUTO_STRING_LENGTH, 64, 70, TRANSPARENT_TEXT);
        Graphics_drawStringCentered(&g_sContext, "Press '*' for rematch",
        AUTO_STRING_LENGTH, 64, 80, TRANSPARENT_TEXT);
        Graphics_flushBuffer(&g_sContext);
    }
}
//Logic for if the game is won and the maximum length is hit
if(j == 10){
     //Paints to the LCD screen
    Graphics_clearDisplay(&g_sContext);
    Graphics_drawStringCentered(&g_sContext, "Congratulations!",
    AUTO_STRING_LENGTH, 64, 50, TRANSPARENT_TEXT);
    Graphics_drawStringCentered(&g_sContext, "You Won!",
    AUTO_STRING_LENGTH, 64, 60, TRANSPARENT_TEXT);
    Graphics_drawStringCentered(&g_sContext, "Press '#' for home",
```

```
                AUTO_STRING_LENGTH, 64, 70, TRANSPARENT_TEXT);
                Graphics_drawStringCentered(&g_sContext, "Press '*' for rematch",
                AUTO_STRING_LENGTH, 64, 80, TRANSPARENT_TEXT);
                Graphics_flushBuffer(&g_sContext);
            }
        } //First for loop
    }// first if
  }  // end infinite loop
}// end main loop

//Gives the countdown at the beginning of the game
void countDown(){
  //Clear, Prints, Refreshes, Delays
  Graphics_clearDisplay(&g_sContext);
  Graphics_drawStringCentered(&g_sContext, "Ready?", AUTO_STRING_LENGTH, 64, 50,
  TRANSPARENT_TEXT);
  Graphics_flushBuffer(&g_sContext);
  swDelay(2);

  //Clear, Prints, Refreshes, Delays
  Graphics_clearDisplay(&g_sContext);
  Graphics_drawStringCentered(&g_sContext, "3...", AUTO_STRING_LENGTH, 64, 50,
```

```
TRANSPARENT_TEXT);
Graphics_flushBuffer(&g_sContext);
swDelay(2);

//Clear, Prints, Refreshes, Delays
Graphics_clearDisplay(&g_sContext);
Graphics_drawStringCentered(&g_sContext, "2...", AUTO_STRING_LENGTH, 64, 50,
TRANSPARENT_TEXT);
Graphics_flushBuffer(&g_sContext);
swDelay(2);

//Clear, Prints, Refreshes, Delays
Graphics_clearDisplay(&g_sContext);
Graphics_drawStringCentered(&g_sContext, "1...", AUTO_STRING_LENGTH, 64, 50,
TRANSPARENT_TEXT);
Graphics_flushBuffer(&g_sContext);
swDelay(2);



//Clear, Prints, Refreshes, Delays
Graphics_clearDisplay(&g_sContext);
```

```
    Graphics_drawStringCentered(&g_sContext, "Go!", AUTO_STRING_LENGTH, 64, 50,
    TRANSPARENT_TEXT);
    Graphics_flushBuffer(&g_sContext);
    swDelay(2);
}

//Assigns the proper values and size to the simon array
unsigned char simonValues(int numLoops, unsigned char *simonArray){
    //Generates values between 1-4
    for (m = 0; m < numLoops; m++) {
      simonArray[m] = (rand() % (4) + 1) + 0x30;
    }
    //Properly 'sizes' the array
    simonArray[numLoops] = '\0';

    //Returns the properly sized array
    return simonArray[numLoops];
}

//Displays each individual simonArray element
void simonDisplay(int numLoops, unsigned char *simonArray){
//Preliminary clearing of the LCD
```

```
Graphics_clearDisplay(&g_sContext);

//Loops through the simonArray to print out the simonValue, refreshes the delay, turns
//the buzzer on and then off for swDelay (1)
for (n= 0; n < numLoops; n++) {
   unsigned char simonVal = simonArray[n];
   if(simonVal == '4') {
      //Paints the character to the LCD
      Graphics_clearDisplay(&g_sContext);
      Graphics_drawStringCentered(&g_sContext, "4", AUTO_STRING_LENGTH, 65, 50,
      TRANSPARENT_TEXT);
      Graphics_flushBuffer(&g_sContext);

      //Turns on and off the buzzer for one second
      BuzzerOn(4);
      swDelay(1);
      BuzzerOff();
   }
   if(simonVal == '3') {
      //Paints the character to the LCD
      Graphics_clearDisplay(&g_sContext);
      Graphics_drawStringCentered(&g_sContext, "3", AUTO_STRING_LENGTH, 65, 50,
```

```
    TRANSPARENT_TEXT);
    Graphics_flushBuffer(&g_sContext);

    //Turns on and off the buzzer for one second
    BuzzerOn(3);
    swDelay(1);
    BuzzerOff();
}
if(simonVal == '2') {
    //Paints the character to the LCD
    Graphics_clearDisplay(&g_sContext);
    Graphics_drawStringCentered(&g_sContext, "2", AUTO_STRING_LENGTH, 65, 50,
    TRANSPARENT_TEXT);
    Graphics_flushBuffer(&g_sContext);

    //Turns on and off the buzzer for one second
    BuzzerOn(2);
    swDelay(1);
    BuzzerOff();
}
if(simonVal == '1') {
    //Paints the character to the LCD
```

```
            Graphics_clearDisplay(&g_sContext);
            Graphics_drawStringCentered(&g_sContext, "1", AUTO_STRING_LENGTH, 65, 50,
            TRANSPARENT_TEXT);
            Graphics_flushBuffer(&g_sContext);

            //Turns on and off the buzzer for one second
            BuzzerOn(1);
            swDelay(1);
            BuzzerOff();
        }
        //Clears the LCD again and then refreshes
        Graphics_clearDisplay(&g_sContext);
        Graphics_flushBuffer(&g_sContext);
    }
}
//This gets rid of the needed delay to wait for a key value by creating an infinite loop for
//checking the keypad if it's pressed.
unsigned char getMyKey(void){
    //returns my key input in a more direct way
    char primer = 0;
```

```
    while(primer == 0){
        unsigned char returnKey = getKey();
        if (returnKey == 0){
        }
        if (returnKey != 0){
            primer = 1;
            return returnKey;
        }
    }
}
void swDelay(char numLoops){
    //This is for the delays within the code.
    //It loops a lot to waste time and takes about a second when an input of 1 is given.
    volatile unsigned int i,j;  // volatile to prevent removal in optimization
                                // by compiler. Functionally this is useless code
    for (j=0; j<numLoops; j++)
    {
        i = 50000 ;             // SW Delay
        while (i > 0)           // could also have used while (i)
          i--;
    }
}
```

Summary and Conclusion:

After meeting the basic requirements of the lab I proceeded to play with the peripherals.h and peripherals.c class. I was able to modify the BuzzerOn() class to take frequency input. After implementing this my code was finished and it was able to hit on all of the requirements for the lab except for the shorter delay as the game goes on. This could be achieved through defining the swDelay constants in the for loops as variables and then adjusting these variable values by a counter within the loop. If this were to be done every two or three moves then one could use modular division for every two moves, and normal division with float values for every three as an indicator if the number is to come out at decimal, the number is not divisible by three. This lab really hit on the points of data types and loop organization. These were the two issues continuously dealt with throughout the lab but by using the debugger I was able to determine the solutions for both issues and create working code for the lab.

Appendices:

1. Lab code attached to submission.
2. Movie file is my code demonstration.