Whack-A-Mole

Oscar Carreon, Ricardo Valverde

EE 128 Mini Project

Section 021

Demo:

https://drive.google.com/file/d/1OBEvPhI6PbMFagEMUVEA7cQKHaOkp-P7/view

Project Description:

Whack-A-Mole is a game in which a mole pops up(LED lights up), and the player whacks the mole(flashes light to a photo resistor) to score. This iteration's win condition is hitting the mole 10 times. This project is implemented on a K64F and Arduino using the C programming language. The project utilizes GPIO to light 4 Leds. ADC to read the voltage change from 4 Photoresistors. SPI to send the players score to the arduino which displays it on an 8x8 Led Matrix. It also utilizes the LEDControl.h library *Copyright © 2007-2015 Eberhard Fahle*

System Design:

As shown in Fig. 1, the K64F begins with a game score of 0. A number is generated at random(1-4) using the built in rand function. The number determines which LED will light up and activates the corresponding PhotoResistor. If the photoresistor picks up a light source in that time frame it will increment the game score and transmit it to the Arduino via SPI serial communication. In Fig. 2 the arduino begins by reading the serial communication port. It is read and stored in the variable score which is displayed on the 8x8 LED Matrix. If the score is higher than 9, the game ends and the LED and LED Matrix play the win sequence.

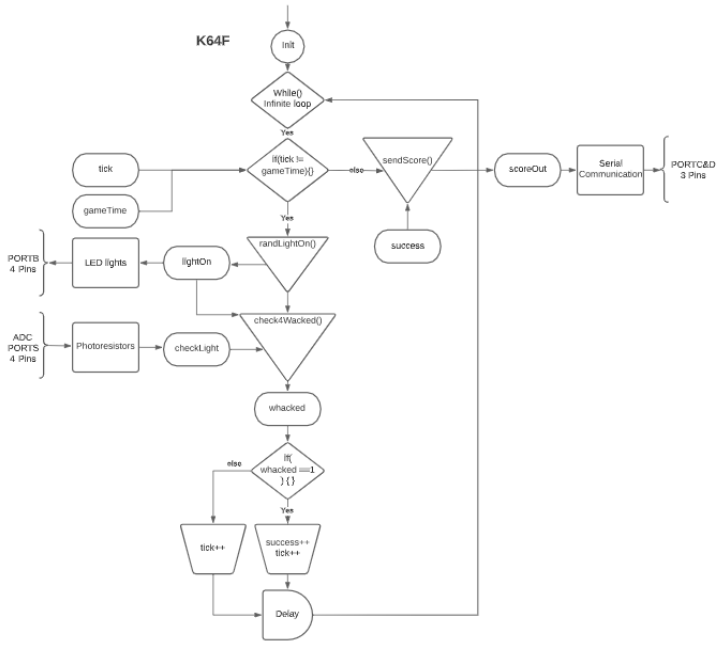


Figure 1: System Flowchart K64F

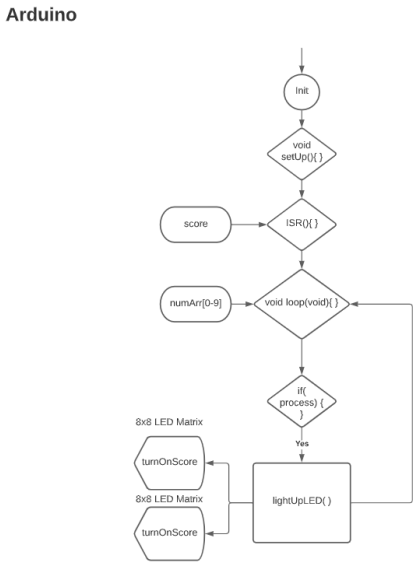
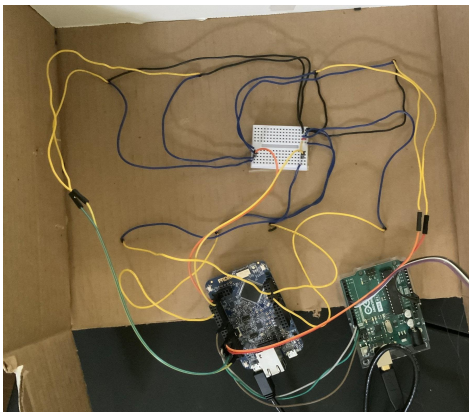


Figure 2: System Flowchart Arduino

Implementation Details:

For the circuit implementation there were a total of 4 LEDs; 4 Photoresistors, 8 Resistors, 1 8x8 LED Matrix. The Final product can be seen in Fig. 3 (a) and 3 (b). On Fig. 4 we show the schematic that displays the topography of this build. For the photoresistor we used PORTS, ADC0_DP0, ADC0_DP1,ADC_SE18, and ADC1_DP0. The LED's were on PORT B and for serial communication we used PORTS C and D.

(a): Whack-A-Mole Setup



(b): Whack-A-Mole Setup Wiring

Figure 3.1: Whack-A-Mole Setup

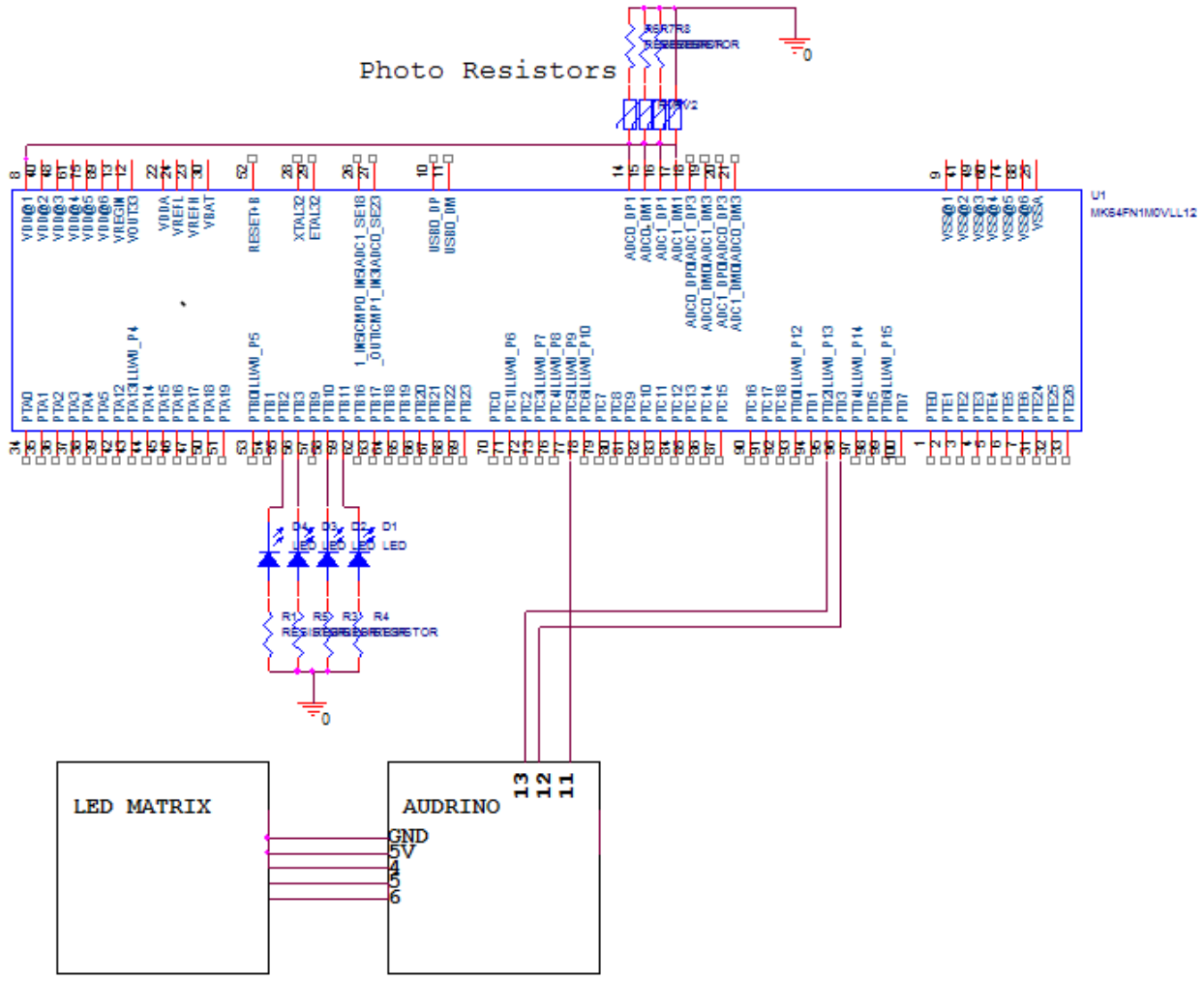


Figure 4 Hardware Schematic

In the kinetis IDE we handled the game logic, controlled the LEDs, and converted the AC signal from the photoresistors. The Arduino received the score of the game and displayed it on a 8x8 LED matrix. The LED logic for the score was implemented using the Arduino IDE. The main portion of the code was in Kinetis and is displayed in Fig. 5 (a) and 5(b).

```
for(int i = 100; i > 0; i--){randNum = rand() % 4 +1;};

if(randNum == 1){
    lightItUp = 0x04;
}
else if (randNum == 2){
    lightItUp = 0x08;
}
else if (randNum == 3){
    lightItUp = 0x200;
}
else if (randNum == 4){
    lightItUp = 0x400;
}
```

(a) Random LED Light Sequence

```
for(int i = 100; i > 0; i--){randNum = rand() % 4 +1;};

if(randNum == 1){
    lightItUp = 0x04;
}
else if (randNum == 2){
    lightItUp = 0x08;
}
else if (randNum == 3){
    lightItUp = 0x200;
}
else if (randNum == 4){
    lightItUp = 0x400;
}
```

(b) Checking if Correct Photoresister is Triggered

Figure 5: Key Portion of Code

Testing/ Evaluating:

This project had three main components consisting of random LED sequence, photoresistor triggering, and serial communication. For the LED sequence we first have to check that we had the correct PORT assignment both in code and physical implementation. Once the code was set up to control the right pins on PORTB, all LEDs were toggled. Having control pins we assigned the random number generator to light up the LEDs. For the photoresistors, a multimeter was used to check that we could physically change the voltage going into the K64F. Using the reference manual we connected the photoresistor to different pins and set up a function for ADC conversion. To check the conversion we set up a watch list in the kinetis IDE. This allowed us to set a threshold value to represent when a photoresistor was triggered. In serial communication we used the Program expert to set up SPI. Iteratively, we sent arbitrary values from K64F to the Arduino. This is how the code was calibrated to send and receive the correct value of the score.

Discussions:

There was an issue on the arduino, when implementing the LED Matrix via the LEDControl library. By itself the Matrix lit up with no issue, when utilizing serial communication, the matrix would flash and stay on a fully lit screen. The issue had to do with a register conflict, as the matrix used SPI on the arduino to receive the data, as well as used to receive score from the K64F. The issue was later resolved by changing the ports that the LED Matrix used. For the photoresistors we had to set up four different ADC pins, this required referencing the manual for the K64F to find how to set up the correct registers and pins. The conversion threshold value took time to figure out but was resolved by using a variable watchlist on the kinetis IDE.

Group Roles:

This project was split into two main departments, Software and hardware. In software development, Ricardo was tasked with creating the game logic for the K64F, setting up serial communication, and displaying the score in the Arduino. Oscar set up the code on the PORTS for the LED and the photoresistors. On the hardware side, Ricardo set up the 8x8LED matrix on the Arduino and the LED for the K64F. Oscar set up the ADC and serial communication pins connection.

Conclusion:

This project served to show the understanding gained from the lab experiments from this course. From the LEDs, it was shown that we were able to have complete control of PORT GPIO registers by initiating a random sequence turning on and off the LEDS. In the lab assignment we were tasked with initiating an ADC port and displaying the value. To show our understanding we increased the level of complexity by setting up multiple ADC pins. The last major component was the SPI communication between the K64F and the Arduino. We took the setup from the lab and changed the code to send the information needed to display the score of the game using the Arduino.

Appendix:

K64F:

```
/**************************************************************************
 * Air Hockey Table Project
 *
 * Author: Oscar Carreon, Ricardo Valverde
 * ------------------------------------------------------------------------
 *          This Code is Created for UCR's EE128 Mini Project
 *                      Wack - A - Mole
 ************************************************************************** /
/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "Pins1.h"
#include "FX1.h"
#include "GI2C1.h"
#include "WAIT1.h"
#include "CI2C1.h"
#include "CsIO1.h"
#include "IO1.h"
#include "MCUC1.h"
#include "SM1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
```

```
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "PDD_Includes.h"
#include "Init_Config.h"
/* User includes (#include below this line is not maintained by Processor Expert) */
#include <stdlib.h>
unsigned char write[512];
// one sec delay
void software_delay(unsigned long delay){
	while(delay>0) delay --;
}
unsigned long Delay = 0x100000;
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
	/************************ ADC PORTS ******************************/
	SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; /*Enable Port A Clock Gate Control*/
	SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK; /*Enable ADC Clock Gate Control*/
	ADC0_CFG1 = 0x0C; //16bits ADC; Bus Clock
	ADC0_SC1A = 0x1F; //Disable the module, ADCH = 11111

	/*****************************************************************************/

/****************************** PORTB ******************************/
	SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;//B
	PORTB_GPCLR = 0x060C0100;//Configure Port B Pins 0-7 for GPIO;
	GPIOB_PDDR = 0x0000060C;//Configure Port B Pins 0-7 for Output;
	// light up LED for testing
	//	GPIOB_PSOR = 0x04;
	//	software_delay(Delay+ (Delay * 4));
	//	GPIOB_PSOR = 0x08;
	//	GPIOB_PSOR = 0x200;
	//	GPIOB_PSOR = 0x400;
/*****************************************************************************/
	/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
	PE_low_level_init();
	/*** End of Processor Expert internal initialization.		***/
	/* Write your code here */
	// Score keeping
	int score = 1;
	// LED being light up
	int lightItUp = 0x00;
	// Photoresistor inputs values
	unsigned int trigger1 = 0;
	unsigned int trigger2 = 0;
```

```
unsigned int trigger3 = 0;
unsigned int trigger4 = 0;
int triggerCheck = 0x00;
int randNum=0;
int win = 0;
int  numCheck = 0;
int len;
LDD_TDeviceData *SM1_DeviceData;
SM1_DeviceData = SM1_Init(NULL);
FX1_Init();
for(;;) {
        /*************************************************
         * game logic psuedocode
         *  rand() = lightItUp;
         *  if(light  up > 0 $ < 5)
         *    pinX = lightitup
         *    check trigger
         *    if only one trigger set triggercheck to led number
         *          if triggercheck == lightitupt
         *          score +=
         *          if score > 0
         *          send score to arduino
         *************************************************/
        for(int i = 100; i > 0; i--){randNum = rand() % 4 +1;};
        if(randNum == 1){
                lightItUp = 0x04;
        }
        else if (randNum == 2){
                lightItUp = 0x08;
        }
        else if (randNum == 3){
                lightItUp = 0x200;
        }
        else if (randNum == 4){
                lightItUp = 0x400;
        }
        if(lightItUp == 0x04){
                GPIOB_PSOR = lightItUp;
                software_delay(Delay*0.5);
                trigger1 = ADC_read16bPH1();
                trigger1 = (trigger1 * 33) / 65535;
        }
        if(lightItUp == 0x08){
                GPIOB_PSOR = lightItUp;
                software_delay(Delay);
```

```
            trigger2 = ADC_read16bPH2();
            trigger2 = (trigger2 * 33) / 65535;
}
if(lightItUp == 0x0200){
            GPIOB_PSOR = lightItUp;
            software_delay(Delay);
            trigger3 = ADC_read16bPH3();
            trigger3 = (trigger3 * 33) / 65535;
}
if(lightItUp == 0x400){
            GPIOB_PSOR = lightItUp;
            software_delay(Delay);
            trigger4 = ADC_read16bPH4();
            trigger4 = (trigger4 * 33) / 65535;
}
if(trigger1 > 12){
            triggerCheck = 0x04;
}
if(trigger2 > 12){
            triggerCheck = 0x08;
}
if(trigger3 > 12){
            triggerCheck = 0x200;
}
if(trigger4 > 12){
            triggerCheck = 0x400;
}
if(triggerCheck == lightItUp){
            score = score + 1;
            if (score > 0) {
                        win = 1 + win;
                        // sending score to Arduino
                        printf("%4d\n",score);
                        len = sprintf(write, "%4d\n",score);
                        SM1_SendBlock(SM1_DeviceData, &write, len);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x060C;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x060C;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0xFFF;
                        software_delay(Delay);
```

```
                        }
                }
/****************** Win LED sequence ***************************/
                if (win == 3) {
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x400;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x04;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x08;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x200;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x400;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x04;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x08;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x200;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x400;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
                        software_delay(Delay*0.5);
                        GPIOB_PSOR = 0x04;
                        software_delay(Delay*0.5);
                        GPIOB_PCOR = 0xFFF;
```

```
                software_delay(Delay*0.5);
                GPIOB_PSOR = 0x08;
                software_delay(Delay*0.5);
                GPIOB_PCOR = 0xFFF;
                software_delay(Delay*0.5);
                GPIOB_PSOR = 0x200;
                software_delay(Delay*0.5);
                GPIOB_PCOR = 0xFFF;
                software_delay(Delay*0.5);
                GPIOB_PSOR = 0x400;
                software_delay(Delay*0.5);

                GPIOB_PCOR = 0xFFF;
                software_delay(Delay*0.5);
                GPIOB_PSOR = 0x60C;
                software_delay(Delay*0.5);
                GPIOB_PCOR = 0xFFF;
                software_delay(Delay*0.5);
                GPIOB_PSOR = 0x60C;
                software_delay(Delay*0.5);
                GPIOB_PCOR = 0xFFF;
                software_delay(Delay*0.5);
                GPIOB_PSOR = 0x60C;
                software_delay(Delay*0.5);
                GPIOB_PCOR = 0xFFF;
                software_delay(Delay*0.5);
                GPIOB_PSOR = 0x60C;
                software_delay(Delay*0.5);
                GPIOB_PCOR = 0xFFF;
                software_delay(Delay*0.5);
                break;
            }
/************************************************************************/
            software_delay(Delay*0.5);
// Reseting all values
            GPIOB_PCOR = 0xFFF;
            lightItUp = 0x00;
            randNum = 0;
            triggerCheck = 0x00;
            software_delay(Delay);
    }
    /* For example: for(;;) { } */

    /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
    /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T
MODIFY THIS CODE!!! ***/
```

```
#ifdef PEX_RTOS_START
        PEX_RTOS_START();               /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
#endif
        /*** End of RTOS startup code.  ***/
        /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
        for(;;){}
        /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

int ADC_read16bPH1(void)// ADCO_PDO
{
        ADC0_SC1A = 0x00; //Write to SC1A to start conversion from ADC_0
        while(ADC0_SC2 & ADC_SC2_ADACT_MASK); //Conversion in progress
        while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); //Until conversion complete
        return ADC0_RA;
}
int ADC_read16bPH2(void) // ADC0_DP1
{
        ADC0_SC1A = 0x01; //Write to SC1A to start conversion from ADC_0
        while(ADC0_SC2 & ADC_SC2_ADACT_MASK); //Conversion in progress
        while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); //Until conversion complete
        return ADC0_RA;
}
int ADC_read16bPH3(void) //ADC_SE18
{
        ADC0_SC1A = 0x13; //Write to SC1A to start conversion from ADC_0
        while(ADC0_SC2 & ADC_SC2_ADACT_MASK); //Conversion in progress
        while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); //Until conversion complete
        return ADC0_RA;
}
int ADC_read16bPH4(void) //ADC1_DP0
{
        ADC0_SC1A = 0x03; //Write to SC1A to start conversion from ADC_0
        while(ADC0_SC2 & ADC_SC2_ADACT_MASK); //Conversion in progress
        while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); //Until conversion complete
        return ADC0_RA;
}
/* END main */
/*!
 ** @}
 */
/*
 ** ###################################################################
 **     This file was created by Processor Expert 10.4 [05.11]
 **     for the Freescale Kinetis series of microcontrollers.
```

```
 ** ###################################################################
 */
```

Arduino:

```
/*********************************************************************************
 * Air Hockey Table Project
 * Author: Oscar Carreon, Ricardo Valverde
 * ------------------------------------------------------------------
 *          This Code is Created for UCR's EE128 Mini Project
 *                      Wack - A - Mole
 *********************************************************************************/
#include <SPI.h>
#include <LedControl.h>
char buff [255];
volatile byte indx;
volatile boolean process;
int score;
int DIN = 6;
int CS =  5;
int CLK = 4;
LedControl lc=LedControl(DIN,CLK,CS,0);
 void setup (void) {
   Serial.begin (115200);
   pinMode(MISO, OUTPUT); // have to send on master in so it set as output
   SPCR |= _BV(SPE); // turn on SPI in slave mode
   indx = 0; // buffer empty
   process = false;
   SPI.attachInterrupt(); // turn on interrupt
   setup1();
}
void setup1(void) {
```

```
// Ledstuff
  lc.shutdown(0,false);
  lc.setIntensity(0,15);      //Adjust the brightness maximum is 15
  lc.clearDisplay(0);
}
ISR (SPI_STC_vect) // SPI interrupt routine
{
  byte c = SPDR; // read byte from SPI Data Register
  if (indx < sizeof(buff)) {
    buff[indx++] = c; // save data in the next index in the array buff
    if (c == '\n') {
      buff[indx - 1] = 0; // replace newline ('\n') with end of string (0)
      process = true;
    }
  }
}
void loop (void) {
   byte *arrayofNum[10];
   // numbers 1 -9
   byte d0[8]=   {0x3C,0x42,0x81,0x81,0x81,0x81,0x42,0x3C};
   byte d1[8]=   {0x03,0x03,0x03,0xFF,0xFF,0xC3,0x63,0x13};

   byte d2[8]= {0x00,0x01,0x79,0xD9,0xCD,0xC7,0x63,0x00};
   byte d3[8]=   {0xDB,0xDB,0xDB,0xDB,0xdB,0x7E,0x7E,0x00};

   byte d4[8]= {0x1F,0x1F,0x10,0x10,0x10,0xFF,0xFF,0x00};
   byte d5[8]= {0x00,0xDF,0xDF,0xDB,0xDB,0xFB,0xF3,0x00};


   byte d6[8]= {0x00,0xFF,0xDF,0xDB,0xDB,0xFB,0xFB,0x00};
```

```
    byte d7[8]= {0x83,0xC3,0xE3,0x3B,0x1F,0x0f,0x03,0x00};



    byte d8[8]= {0x00,0x76,0x99,0x99,0x99,0x99,0x76,0x00};
    byte d9[8]= {0x3F,0x33,0x33,0x33,0x33,0xFF,0xFF,0x00};

    //Win flash
    byte w1[8]= {0xAB,0xD5,0xAB,0xD5,0xAB,0xD5,0xAB,0xD5};
    byte w2[8]= {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};
    byte w3[8]= {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

    arrayofNum[0] = d0;
    arrayofNum[1] = d1;
    arrayofNum[2] = d2;
    arrayofNum[3] = d3;
    arrayofNum[4] = d4;
    arrayofNum[5] = d5;
    arrayofNum[6] = d6;
    arrayofNum[7] = d7;
    arrayofNum[8] = d8;
    arrayofNum[9] = d9;
// Checks information was received
   if (process) {
    score = 1 + score;
      process = false; //reset the process
      Serial.println("score");
      Serial.println(score);
      // Debuggin code
//      Serial.println (buff); //print the array on serial monitor
      indx= 0; //reset button to zero
```

```
  }
  /// Debugging code
//   Serial.println("outside loop");
//   Serial.println(score);
if (score < 3){
   delay(100);

   printByte(arrayofNum[score]);
   delay(100);
 }

 else {
   delay(50);
   printByte(w1);
   delay(100);
   printByte(w2);
   delay(100);
   printByte(w3);
   delay(100);
 }
}

void printByte(byte character [])
{
 int i = 0;
 for(i=0;i<8;i++)
 {
   lc.setRow(0,i,character[i]);
 }
}
```