Introduction to Embedded Systems Final Project
Reflex Game

Yusuf Serdar Çalışkan - 2018513018 - 12.01.2021

Introduction:

This is a game with 2 players with 2 buttons for each player. Game starts when ‘ * ‘ symbol is shown in the lcd. After 1 second, green or red LED will be turned on randomly. First player to push the correct button wins the round. If any player presses the wrong button (including after pressing the correct one) that player loses the round. Pressing any key before the leds getting turned on or not pressing any at all will also end up with loss.

2 seconds after LEDs get turned on they will get closed and scoring will be shown. The rounds will repeat themself until there is a valid winner. Total of rounds that will be played is determined by the *gameCount* variable (default 3). After reaching the end of all rounds, end game scoring will be shown with player scores and best response times. 10 seconds later the game will restart itself.

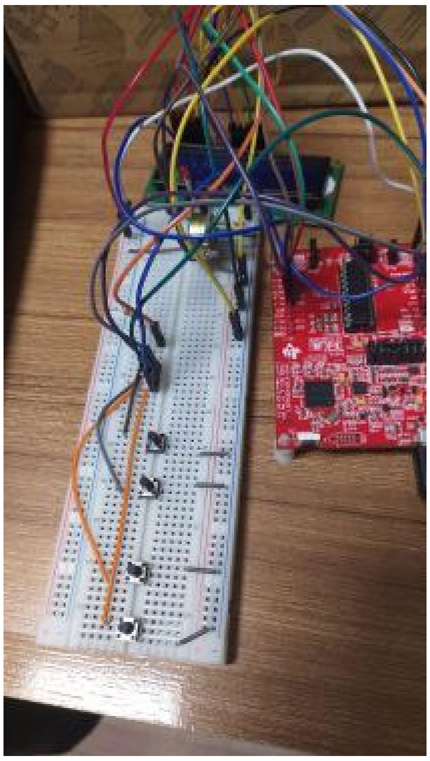
Materials:

- 4 *button*
- 1 *16x2 lcd*
- *msp430g2553*
- 1 *green led*
- 1 *red led*
- M-F, M-M, F-F jumper cables
- 1 *breadboard*
- 1 *10k potentiometer*

Connections:

P2.0 -> LCD D4
P2.1 -> LCD D5
P2.2 -> LCD D6
P2.3 -> LCD D7
P2.4 -> LCD E
P2.5 -> LCD RS

P1.1 -> PLAYER 1 green BUTTON
P1.2 -> PLAYER 1 red BUTTON
P1.3 -> PLAYER 2 green BUTTON
P1.4 -> PLAYER 2 red BUTTON
Cross-section of buttons to GND
5 VOLT to breadboard
GRD to breadboard
LCD RW to GND
LCD D0 to D3 no connection
LCD VSS to GND
LCD VDD to 5V
LCD V0 to 5V through 10K potentiometer
P1.7 -> green LED
P1.6 -> red LED

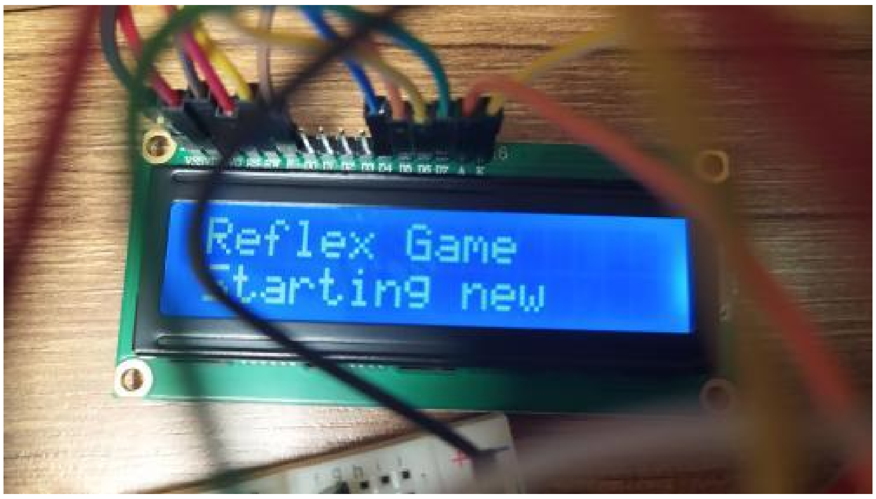


(from top to bottom buttons are P2red, P2 green, P1green, P1 red)

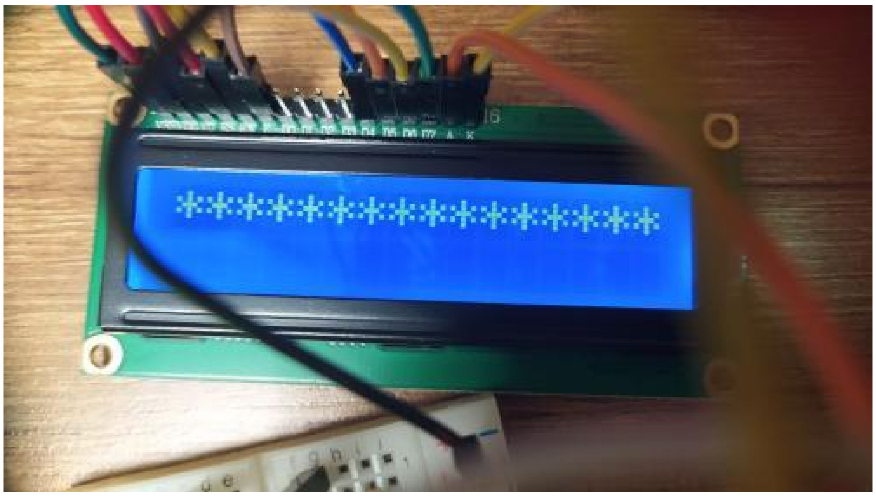
How it works:

When first booted required libraries will be included, definitions will be made. Required configurations will be done. Details can be found in comment sections in the code.

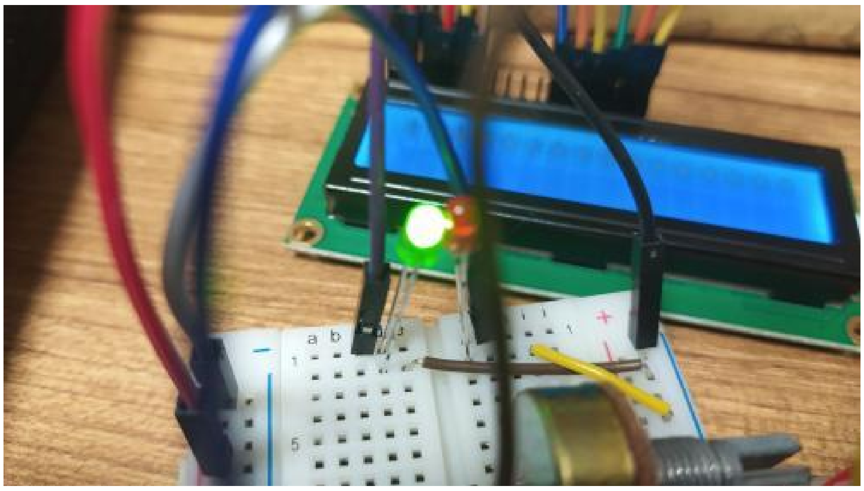
After boot and every time a new game initiates this notification will be shown in the lcd for 3 seconds, giving time to players in order to prepare themself.



‘ * ‘ symbol will be shown in the lcd and the game will be started. Now if any player presses a button they will be disqualified as they pressed before any LED gets turned on.



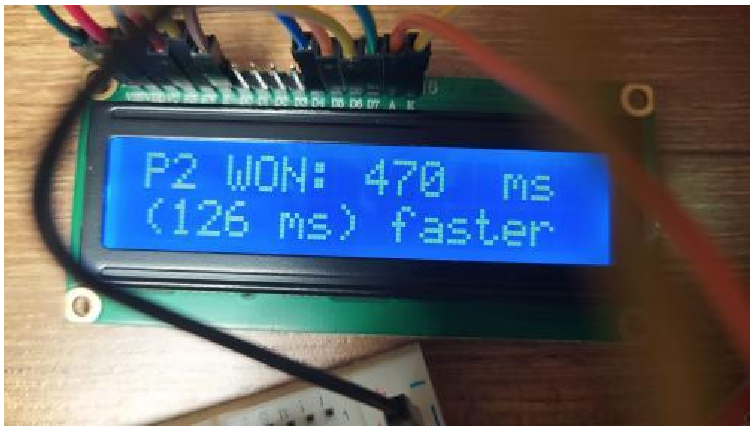
1 second later a random number will be generated which turns on the green or the red LED. Now if a player presses the correct button corresponding to the color of LED their score time will be saved. If any player presses the wrong button until LEDs get turned off they will be disqualified no matter what. Leds will get turned off after staying on for 2 seconds. Any player who didn’t press any button will be marked as missed.



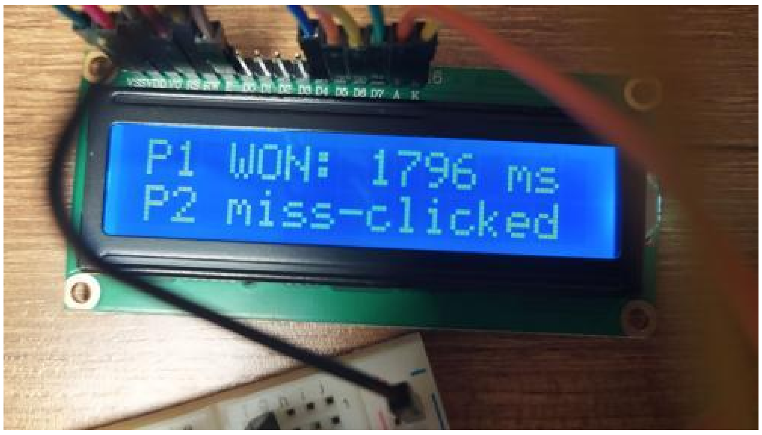
After the LED gets turned off, scoring will start. There are 4 possible situations for players.

1. They pressed correct
2. They pressed wrong
3. They pressed early
4. They didn't pressed

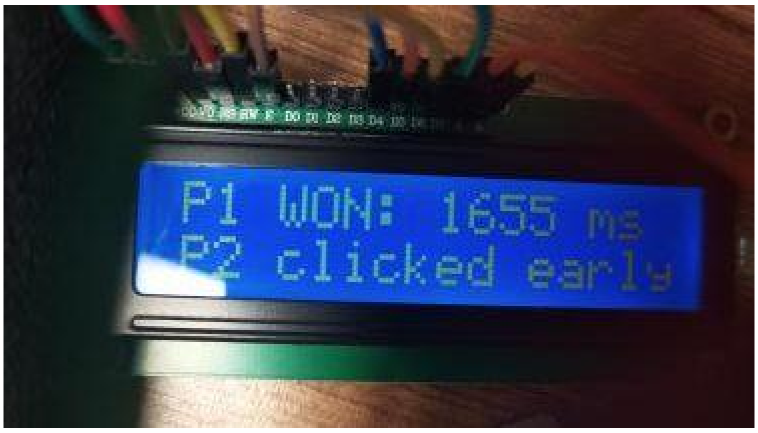
For each 16 (+2) situation different displays are ready to be shown. Some of them are pictured below. If there is a valid winner "*gameCount*" will be reduced by one. Scoring will be shown for 4 seconds before new round starts



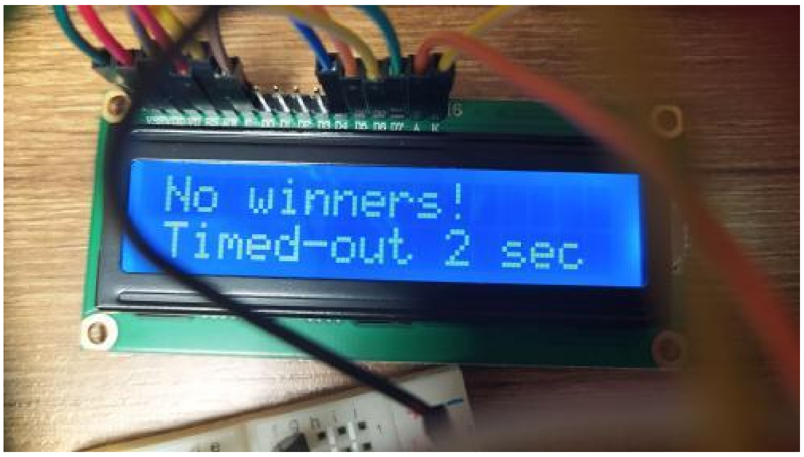
```
P2 WON: 470  ms
(126 ms) faster
```



```
P1 WON: 1796 ms
P2 miss-clicked
```

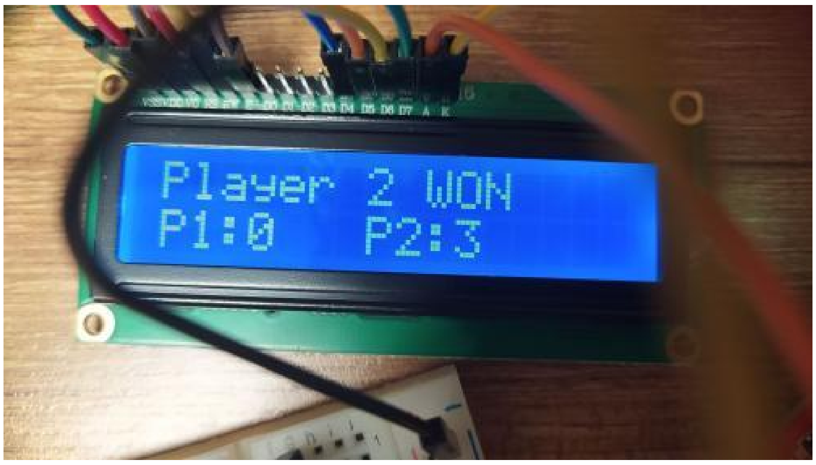


```
P1 WON: 1655 ms
P2 clicked early
```



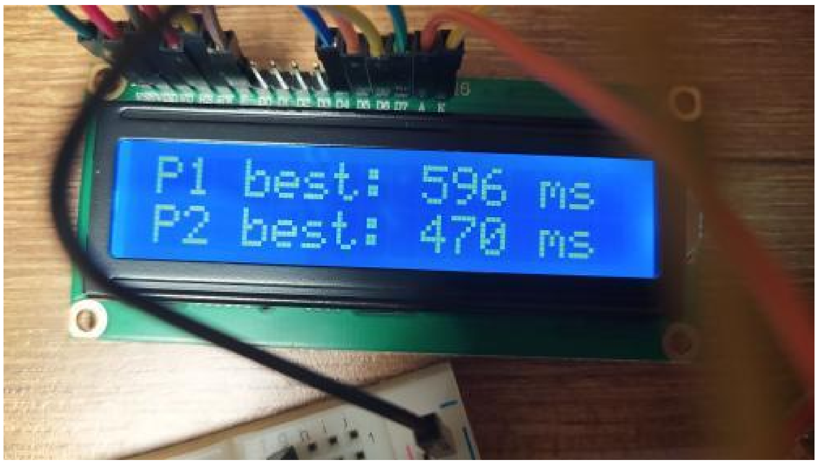
```
No winners!
Timed-out 2 sec
```

The game will continue until there have been 3 valid rounds with a winner. After the last round scoring, for 5 seconds the winner and round scoring will be shown.



```
Player 2 WON
P1:0    P2:3
```

Then for 5 seconds best response times for each player for that game will be shown.

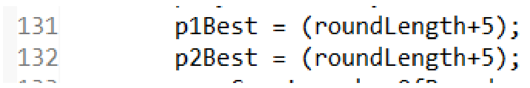


```
P1 best: 596 ms
P2 best: 470 ms
```

A new game will be initiated.

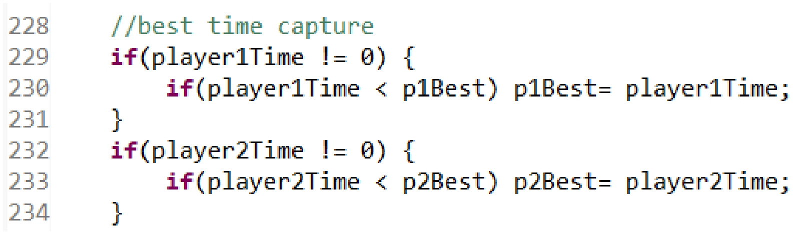
Feature: Best response times

This is an additional feature which wasn't requested in the original introduction. At the start of the game the players best response time variable will be set to lengthOfRound+5.



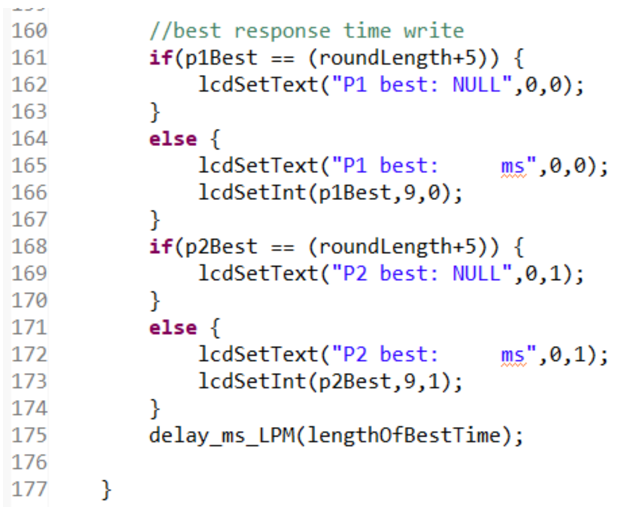
```
        p1Best = (roundLength+5);
        p2Best = (roundLength+5);
```

At the end of each round if a player only pressed the correct button and it's lower than their best response time, round timing will be written over the best response time variable.



```
    //best time capture
    if(player1Time != 0) {
        if(player1Time < p1Best) p1Best= player1Time;
    }
    if(player2Time != 0) {
        if(player2Time < p2Best) p2Best= player2Time;
    }
```

At the end of the game best response time will be written



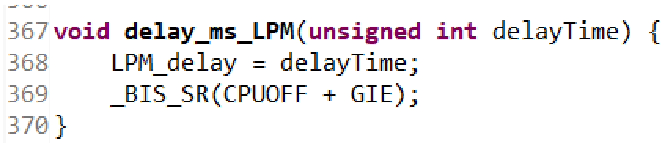
```
        //best response time write
        if(p1Best == (roundLength+5)) {
            lcdSetText("P1 best: NULL",0,0);
        }
        else {
            lcdSetText("P1 best:     ms",0,0);
            lcdSetInt(p1Best,9,0);
        }
        if(p2Best == (roundLength+5)) {
            lcdSetText("P2 best: NULL",0,1);
        }
        else {
            lcdSetText("P2 best:     ms",0,1);
            lcdSetInt(p2Best,9,1);
        }
        delay_ms_LPM(lengthOfBestTime);

    }
```

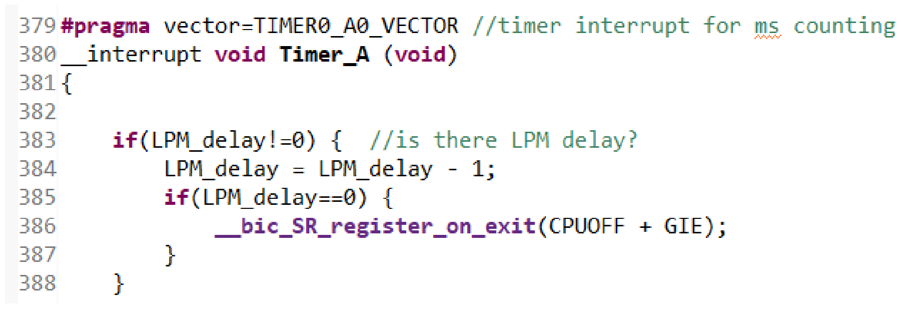
Feature: Low power mode delays

This was an experimental feature which made into final build half-finished. Originally I intended to make the LPM delay function by changing CCR0 to its maximum value so it can stay in LPM as much as it can but didn't have enough time to detail this. So I just left the experimental code in the final-build.

When the function is called MCU is put in LPM0 mode and will wake up every 1ms to check if delay time is achieved.



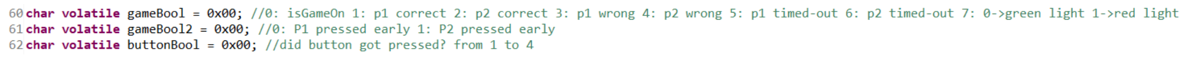
```
void delay_ms_LPM(unsigned int delayTime) {
    LPM_delay = delayTime;
    _BIS_SR(CPUOFF + GIE);
}
```



```c
379 #pragma vector=TIMER0_A0_VECTOR //timer interrupt for ms counting
380 __interrupt void Timer_A (void)
381 {
382 
383     if(LPM_delay!=0) {  //is there LPM delay?
384         LPM_delay = LPM_delay - 1;
385         if(LPM_delay==0) {
386             __bic_SR_register_on_exit(CPUOFF + GIE);
387         }
388     }
```

Feature: Booleans

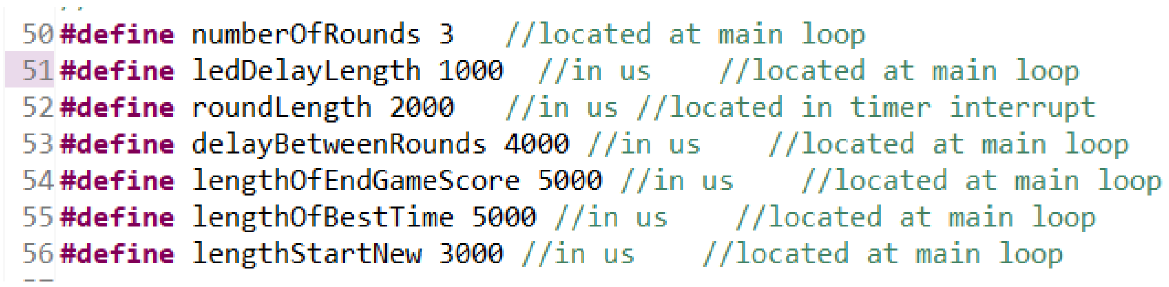
From my research msp430g2553 doesn't support boolean variables even with the stdbool.h library included. So I have made my own boolean structure shown as below and worked with them using bitwise operations. All done in order to save from memory.



```c
60 char volatile gameBool = 0x00; //0: isGameOn 1: p1 correct 2: p2 correct 3: p1 wrong 4: p2 wrong 5: p1 timed-out 6: p2 timed-out 7: 0->green light 1->red light
61 char volatile gameBool2 = 0x00; //0: P1 pressed early 1: P2 pressed early
62 char volatile buttonBool = 0x00; //did button got pressed? from 1 to 4
```

Feature: Settings block

For easier changes in the game I have included settings block at the very top of my code. This is memory friendly as it uses #define instead of creating a variable.



```c
50 #define numberOfRounds 3   //located at main loop
51 #define ledDelayLength 1000  //in us    //located at main loop
52 #define roundLength 2000   //in us //located in timer interrupt
53 #define delayBetweenRounds 4000 //in us    //located at main loop
54 #define lengthOfEndGameScore 5000 //in us    //located at main loop
55 #define lengthOfBestTime 5000 //in us    //located at main loop
56 #define lengthStartNew 3000 //in us    //located at main loop
```

numberOfRounds: Changes how many rounds it will take to reach end of the game.
ledDelayLength: For how long ' * ' symbol will shown without LEDs getting turned on.
roundLength: For how long LEDs will stay turned on.
delayBetweenRounds: For how long end-round scoring will be shown.
lengthOfEndGameScore: For how long end-game winner announcement will be shown.
lengthOfBestTime: For how long best time responses will be shown.
lengthStartNew: For how long "Starting new" text will be shown at the start of the games.

Note: Since i used 1Mhz clock writing time in us is ok. But should be changed if any other clock timing is used.

Feature: Interrupts

For both delays, time counting and button inputs interrupts are used.
For timer interrupts please see line379 in the code.
For button interrupts please see line414 in the code.

Note: If the clock is changed then CCR0 should be re-set to 1ms.

Note: lcdLib

lcdLib is an online found library for 16x2 lcd which uses pins from P2.0 to P2.5. In case of clock change delay times should be re-set in lcdLib.h

Source for lib: http://www.ece.utep.edu/courses/web3376/Lab_5_-_LCD.html

Code:

(for easier readability and access to main code and library code a private github repository has been created. For access please email to "yserdarcaliskan@gmail.com", github link: https://github.com/Sravdar/msp430-reflex-game)

```
/*
Name: Reflex Game
Author: Yusuf Serdar Caliskan
Started at date: 26.12.2021
Last update: 6.01.2022

CONNECTIONS
P2.0 -> LCD D4
P2.1 -> LCD D5
P2.2 -> LCD D6
P2.3 -> LCD D7
P2.4 -> LCD E
P2.5 -> LCD RS
P1.0 -> NULL (reserved for embedded red light)
P1.1 -> PLAYER 1 green BUTTON
P1.2 -> PLAYER 1 red BUTTON
P1.3 -> PLAYER 2 green BUTTON
P1.4 -> PLAYER 2 red BUTTON
P1.6 -> NULL (reserved for embedded green light)
3.3 VOLT to breadboard
GRD to breadboard
RW to GRND, D0 to D3 no connection
P1.7 -> green led (p1.0 for internal light then change related parts bellow in code. search
"internal" to find)
P1.6 -> red led


Notes to self:
1_partialy done:  creating a new delay function which puts MCU in LPM could be great, extra
features!



*/
#include <msp430g2553.h>
#include "lcdLib.h" //lcd lib from: http://www.ece.utep.edu/courses/web3376/Lab_5_-_LCD.html
#include <stdlib.h> //lib for random number and abs()
#include <time.h>   //lib for random number

//DISABLED SINCE I DONT USE THESE INSTEAD USE LPM DELAY FUNCTION I WROTE
```

```
//#define delay_s(x)     __delay_cycles((long) x* 1000000)
//#define delay_ms(x)     __delay_cycles((long) x* 1000)
//#define delay_s(x)     __delay_cycles(1)

void game();    //main game function which is 1 round
void writeScores(); //to display scores of round
void delay_ms_LPM(unsigned int);    //low power delay mode which wakes up every 1 ms


//******
//**********SETTINGS**********
#define numberOfRounds 3   //located at main loop
#define ledDelayLength 1000 //in us    //located at main loop
#define roundLength 2000   //in us //located in timer interrupt
#define delayBetweenRounds 4000 //in us    //located at main loop
#define lengthOfEndGameScore 5000 //in us    //located at main loop
#define lengthOfBestTime 5000 //in us    //located at main loop
#define lengthStartNew 3000 //in us    //located at main loop


unsigned volatile int scoreTime = 0;
char volatile gameBool = 0x00; //0: isGameOn 1: p1 correct 2: p2 correct 3: p1 wrong 4: p2 wrong 5:
p1 timed-out 6: p2 timed-out 7: 0->green light 1->red light
char volatile gameBool2 = 0x00; //0: P1 pressed early 1: P2 pressed early
char volatile buttonBool = 0x00; //did button got pressed? from 1 to 4
unsigned short gameCount = 0;
unsigned volatile int player1Time;
unsigned volatile int player2Time;
unsigned volatile int player1Score;
unsigned volatile int player2Score;
unsigned volatile int p1Best;
unsigned volatile int p2Best;
unsigned int randomNum;
unsigned volatile int LPM_delay = 0; //low power mode delay (wakes up every 1ms)






/**
 * main.c
 */
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    //Clock settings
    BCSCTL1 = CALBC1_1MHZ;                    // 1MHz clock
    DCOCTL = CALDCO_1MHZ;
    IFG1 &= ~OFIFG; // Set Interrupt Flag Generator
    BCSCTL2 |= SELM_1;



    //Timer A control register
    TACTL = TASSEL_2 + MC_1;           // SMCLK, upmode //_BIS_SR(GIE); to enable interrupts
    CCR0 =  1000;
    CCTL0 = CCIE;                      // CCR0 interrupt enabled

    //LCD outputs
    P2DIR |= BIT0 + BIT1 + BIT2 + BIT3+ BIT4+ BIT5;
    //LED outputs
    P1DIR |=  BIT6;
    //P1DIR |= BIT0;      //enable for internal light
    //P1REN |= BIT0;      //enable for internal light (without this it doesn't shine. probably noise
from timerA)
    P1DIR |= BIT7;        //disable for internal light

    //BUTTON CONFIG
    P1DIR &= ~BIT1 + ~BIT2 + ~BIT3 + ~BIT4; //buttons as input
    P1REN |= BIT1 + BIT2 + BIT3 + BIT4;
    P1IE |= BIT1 + BIT2 + BIT3 + BIT4;  //enable interrupts
    P1IES |= BIT1 + BIT2 + BIT3 + BIT4; //high to low selection
    P1OUT |= BIT1 + BIT2 + BIT3 + BIT4;
    P1OUT &= ~BIT6;
    //P1OUT  &= ~BIT0;  //re enable for internal led
    P1OUT  &= ~BIT7;    //disable for internal led
    P1IFG &= ~BIT1 + ~BIT2 + ~BIT3 + ~BIT4;

    srand(time(NULL)); //random intiallazion


    lcdInit();  //gotta run once to intialize 16x2 lcd

    //MAIN LOOP
    while(1) {
        lcdClear();
        lcdSetText("Reflex Game",0,0);
        lcdSetText("Starting new",0,1);

        player1Score=0;
        player2Score=0;
        p1Best = (roundLength+5);   //this is done so i can compare and change it while assigning it
and don't write it if player didn't pressed at all
        p2Best = (roundLength+5);   //^without using and other variable
        gameCount=numberOfRounds;    //game count set

        delay_ms_LPM(lengthStartNew);

        //ROUNDS
        while(gameCount != 0) { //main game loop, will call the function every "delayBetweenRounds"
until gameCount fulfilled. note: only rounds with valid
            game();             //^winners will reduce the gameCount
            delay_ms_LPM(delayBetweenRounds);
        }

        //game end scoring
        lcdClear();
        if(player1Score>player2Score) {
            lcdSetText("Player 1 WON",0,0);
            lcdSetText("P1:    P2: ",0,1);
            lcdSetInt(player1Score,3,1);
            lcdSetInt(player2Score,10,1);
        }
        else if(player1Score<player2Score) {
            lcdSetText("Player 2 WON",0,0);
            lcdSetText("P1:    P2: ",0,1);
            lcdSetInt(player1Score,3,1);
            lcdSetInt(player2Score,10,1);
        }
        delay_ms_LPM(lengthOfEndGameScore);


        //best response time write
```

```c
        if(p1Best == (roundLength+5)) {
            lcdSetText("P1 best: NULL",0,0);
        }
        else {
            lcdSetText("P1 best:     ms",0,0);
            lcdSetInt(p1Best,9,0);
        }
        if(p2Best == (roundLength+5)) {
            lcdSetText("P2 best: NULL",0,1);
        }
        else {
            lcdSetText("P2 best:     ms",0,1);
            lcdSetInt(p2Best,9,1);
        }
        delay_ms_LPM(lengthOfBestTime);

    }


}









void game() {
    lcdClear();
    lcdSetText("****************",0,0);
    randomNum=3;    //to punish those pressed before led
    gameBool |= BIT0; //gameOn    (to start game before led disable bellow, enable this line)
    _BIS_SR(GIE);   //interrupts enabled

    delay_ms_LPM(ledDelayLength);
    _BIS_SR(GIE);//re enable after delay closes it


    //RANDOM LED*********
    randomNum = rand()%2;   //1=green
    if(randomNum==0) {
    //if(1) {
        gameBool &= ~BIT7;//green
        //P1OUT |= BIT0;    //enable for internal led
        P1OUT |= BIT7;  //disable for internal led
    }
    else {
        gameBool |= BIT7;//red
        P1OUT |= BIT6;
    }



    scoreTime=0;
    //gameBool |= BIT0; //gameOn    (to start game after led disable above, enable this line)

    while((gameBool & BIT0)==BIT0) {
        _BIS_SR(CPUOFF);  //LMP enabled
    }

    //close leds
    //P1OUT &= ~BIT0;   //enable for internal led
    P1OUT &= ~BIT7;     //disable for internal led
    P1OUT &= ~BIT6;

    writeScores();  //writes times to lcd and reducts gameCount if there's a winner

    //best time capture
    if(player1Time != 0) {
        if(player1Time < p1Best) p1Best= player1Time;
    }
    if(player2Time != 0) {
        if(player2Time < p2Best) p2Best= player2Time;
    }

    //Re-set variables
    gameBool = 0x00;
    gameBool2 = 0x00;
    buttonBool = 0x00;
    scoreTime = 0;
    player1Time=0;
    player2Time=0;
}





void writeScores() {   //write scores of ROUND
    lcdClear();
    if( ((gameBool & BIT1)!=0) && ((gameBool & BIT2)!=0)  ) { //P1 correct P2 correct
        if(player2Time==player1Time) {
            lcdSetText("WOW, ITS DRAW",0,0);    //not sure if this is even possible
            lcdSetText("Scored:      ms",0,1);
            lcdSetInt(player1Time,8,1);
        }
        else if(player2Time>player1Time) {
            lcdSetText("P1 WON:      ms ",0,0);
            lcdSetText("(    ms) faster",0,1);
            lcdSetInt(player1Time,8,0);
            lcdSetInt( abs(player1Time-player2Time) ,1,1);
            player1Score++;
            gameCount--;
        }
        else if(player2Time<player1Time) {
            lcdSetText("P2 WON:      ms ",0,0);
            lcdSetText("(    ms) faster",0,1);
            lcdSetInt(player2Time,8,0);
            lcdSetInt( abs(player2Time-player1Time) ,1,1);
            player2Score++;
            gameCount--;
        }
    }
    else if( ((gameBool & BIT1)!=0) && ((gameBool & BIT4)!=0)  ) {    //P1 correct P2 wrong
        lcdSetText("P1 WON:      ms ",0,0);
        lcdSetText("P2 miss-clicked",0,1);
        lcdSetInt(player1Time,8,0);
        player1Score++;
        gameCount--;
    }
    else if( ((gameBool & BIT2)!=0) && ((gameBool & BIT3)!=0)  ) {    //P1 wrong P2 correct
        lcdSetText("P2 WON:      ms ",0,0);
        lcdSetText("P1 miss-clicked",0,1);
        lcdSetInt(player2Time,8,0);
        player2Score++;
        gameCount--;
    }
    else if( ((gameBool & BIT1)!=0) && ((gameBool & BIT6)!=0)  ) {    //P1 correct P2 timed-out
        lcdSetText("P1 WON:      ms ",0,0);
        lcdSetText("P2 timed-out",0,1);
        lcdSetInt(player1Time,8,0);
        player1Score++;
        gameCount--;
    }
    else if( ((gameBool & BIT2)!=0) && ((gameBool & BIT5)!=0)  ) {    //P1 timed-out P2 correct
        lcdSetText("P2 WON:      ms ",0,0);
        lcdSetText("P1 timed-out",0,1);
        lcdSetInt(player2Time,8,0);
        player2Score++;
        gameCount--;
    }
    else if( ((gameBool & BIT5)!=0) && ((gameBool & BIT6)!=0)  ) {    //both timed-out
        lcdSetText("No winners!",0,0);
        lcdSetText("Timed-out 2 sec",0,1);
    }
    else if( ((gameBool & BIT3)!=0) && ((gameBool & BIT6)!=0)  ) {    //p1 wrong p2 timed-out
        lcdSetText("P1 pressed wrong",0,0);
        lcdSetText("P2 timed-out",0,1);
    }
    else if( ((gameBool & BIT5)!=0) && ((gameBool & BIT4)!=0)  ) {    //p1 timed-out p2 wrong
        lcdSetText("P2 pressed wrong",0,0);
        lcdSetText("P1 timed-out",0,1);
    }
    else if( ((gameBool & BIT3)!=0) && ((gameBool & BIT4)!=0)  ) {    //both wrong
        lcdSetText("Both players",0,0);
        lcdSetText("pressed wrong",0,1);
    }
    else if( ((gameBool & BIT1)!=0) && ((gameBool2 & BIT1)!=0)  ) {    //p1 correct, p2 pressed
early
        lcdSetText("P1 WON:      ms ",0,0);
        lcdSetText("P2 clicked early",0,1);
        lcdSetInt(player1Time,8,0);
        player1Score++;
        gameCount--;
    }
    else if( ((gameBool & BIT2)!=0) && ((gameBool2 & BIT0)!=0)  ) {    //p2 correct, p1 pressed
early
        lcdSetText("P2 WON:      ms ",0,0);
        lcdSetText("P1 clicked early",0,1);
        lcdSetInt(player2Time,8,0);
        player2Score++;
        gameCount--;
    }
    else if( ((gameBool & BIT3)!=0) && ((gameBool2 & BIT1)!=0)  ) {    //p1 wrong, p2 pressed early
        lcdSetText("P1 pressed wrong",0,0);
        lcdSetText("P2 clicked early",0,1);
    }
    else if( ((gameBool & BIT4)!=0) && ((gameBool2 & BIT0)!=0)  ) {    //p2 wrong, p1 pressed early
        lcdSetText("P1 clicked early",0,0);
        lcdSetText("P2 pressed wrong",0,1);
    }
    else if( ((gameBool & BIT5)!=0) && ((gameBool2 & BIT1)!=0)  ) {    //p1 timed out, p2 pressed
early
        lcdSetText("P1 timed-out",0,0);
        lcdSetText("P2 clicked early",0,1);
    }
    else if( ((gameBool & BIT6)!=0) && ((gameBool2 & BIT0)!=0)  ) {    //p2 timed out, p1 pressed
early
```

```
        lcdSetText("P1 clicked early",0,0);
        lcdSetText("P2 timed-out",0,1);
    }
    else if( ((gameBool2 & BIT1)!=0) && ((gameBool2 & BIT0)!=0)  ) {    //both pressed early
        lcdSetText("Both players",0,0);
        lcdSetText("pressed early",0,1);
    }
    else {  //added in case something went wrong above, might delete this as it stands useless since
i wrote all possible 16 situations above
        lcdSetText("Invalid result",0,0);
        lcdSetText("Starting new",0,1);
    }


}








void delay_ms_LPM(unsigned int delayTime) { //low power delay function. not so optimized as it wakes
up every 1ms, can improve this actually to be
    LPM_delay = delayTime;                  //^low power delay function. could be done by changing
CCR0 but gotta look more into it.
    _BIS_SR(CPUOFF + GIE);
}








#pragma vector=TIMER0_A0_VECTOR //timer interrupt for ms counting
__interrupt void Timer_A (void)
{

    if(LPM_delay!=0) {  //is there LPM delay?
        LPM_delay = LPM_delay - 1;
        if(LPM_delay==0) {
            __bic_SR_register_on_exit(CPUOFF + GIE);
        }
    }


    if( ((gameBool & BIT0)!=0)&& (randomNum!=3) ) {
        scoreTime++;
       if(scoreTime>roundLength) {
           if( ((gameBool & BIT1)==0)&&((gameBool & BIT3)==0)&&((gameBool2 & BIT0)==0)   ) { //did
player 1 timed-out
               gameBool |= BIT5;
           }
           if( ((gameBool & BIT2)==0)&&((gameBool & BIT4)==0)&&((gameBool2 & BIT1)==0)   ) { //did
player 2 timed-out
               gameBool |= BIT6;
           }
           gameBool &= ~BIT0;
           __bic_SR_register_on_exit(CPUOFF + GIE);  //LMP and interrupts disabled

       }
    }
}








#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    //led is off
    if(randomNum==3) {
        if( ((P1IN & BIT1) == 0) || ((P1IN & BIT2) == 0) ) {    //if player 1 is early
            gameBool2 |= BIT0;
            buttonBool |= BIT1;
            buttonBool |= BIT2;
        }
        if( ((P1IN & BIT3) == 0) || ((P1IN & BIT4) == 0) ) {    //if player 1 is early
            gameBool2 |= BIT1;
            buttonBool |= BIT3;
            buttonBool |= BIT4;
        }
    }


    //led is on
    if((gameBool & BIT0)== BIT0  ) { //is game on?
        if( ((P1IN & BIT1) == 0) && ((buttonBool & BIT1) == 0) ) {    //p1 green
            if( (gameBool&BIT7) == BIT7  ) {    //game lose
                gameBool |= BIT3;
                gameBool &= ~BIT1;
                player1Time = 0;
                buttonBool |= BIT2;
            }
            else if((gameBool&BIT3)==0){   //game win
                gameBool |= BIT1;
                player1Time = scoreTime;
            }
            buttonBool |= BIT1;
        }

        if( ((P1IN & BIT2) == 0) && ((buttonBool & BIT2) == 0) ) {    //p1 red
            if( (gameBool&BIT7) != BIT7  ) {    //game lose
                gameBool |= BIT3;
                gameBool &= ~BIT1;
                player1Time = 0;
                buttonBool |= BIT1;
            }
            else if((gameBool&BIT4)==0){   //game win
                gameBool |= BIT1;
                player1Time = scoreTime;
            }
            buttonBool |= BIT2;
        }

        if( ((P1IN & BIT3) == 0) && ((buttonBool & BIT3) == 0) ) {    //p2 green
            if( (gameBool&BIT7) == BIT7  ) {    //game lose
                gameBool |= BIT4;
                gameBool &= ~BIT2;
                player2Time = 0;
                buttonBool |= BIT4;
            }
            else{   //game win
                gameBool |= BIT2;
                player2Time = scoreTime;
            }
            buttonBool |= BIT3;
        }

        if( ((P1IN & BIT4) == 0) && ((buttonBool & BIT4) == 0) ) {    //p2 red
            if( (gameBool&BIT7) != BIT7  ) {    //game lose
                gameBool |= BIT4;
                gameBool &= ~BIT2;
                player2Time = 0;
                buttonBool |= BIT3;
            }
            else{   //game win
                gameBool |= BIT2;
                player2Time = scoreTime;
            }
            buttonBool |= BIT4;
        }
    }


    P1IFG = 0x00;
}
```