CS3514

Assignment 1

Two-Person Reaction Timer Game

Author: Alberto García Hernández - 118105946

Date: (13/11/2018)

Table of Contents

1. INTRODUCTION

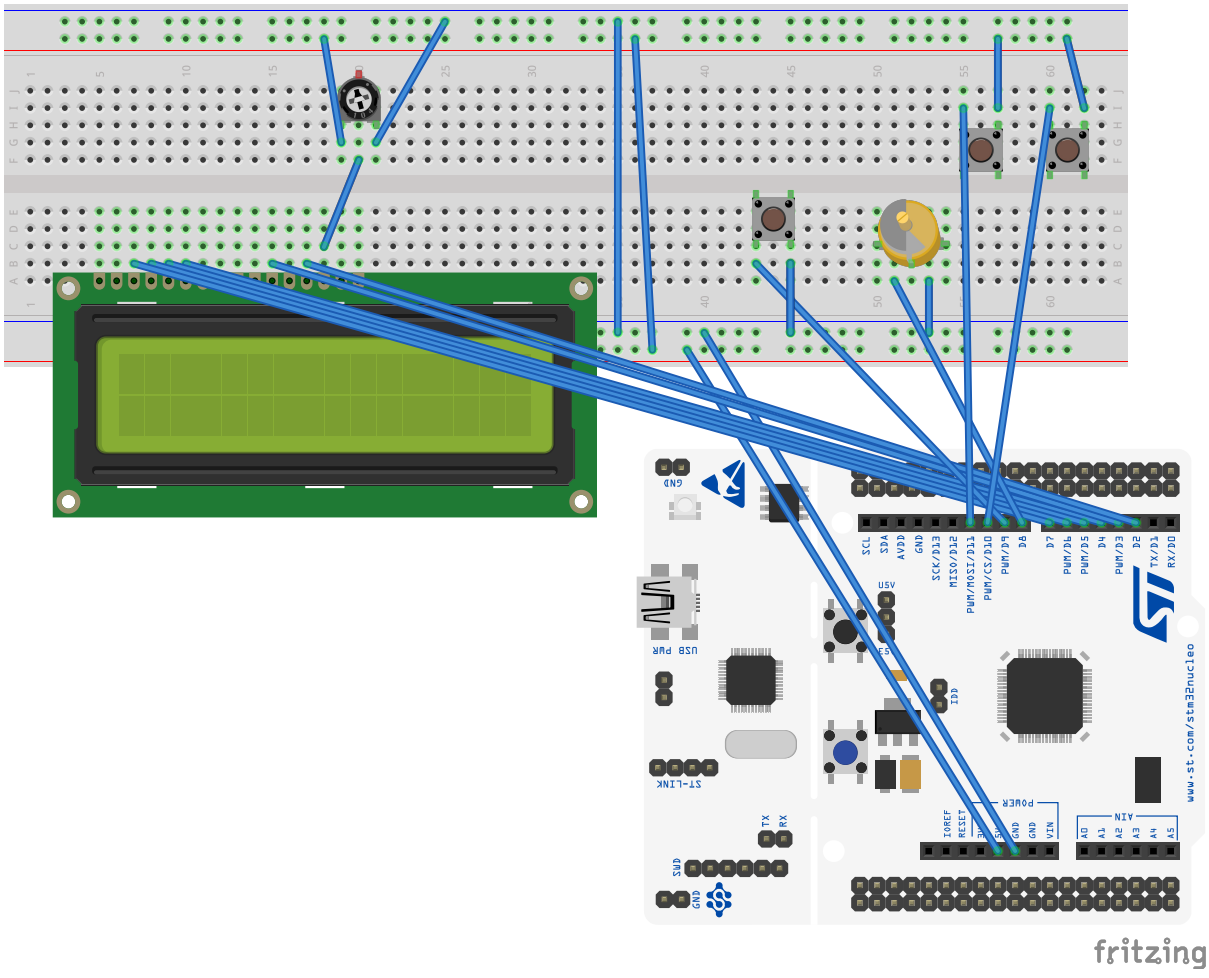
The objective of this assignment is to implement a reaction timer game, where each player has too pressed their respective button as fast as possible after the buzzer sounds, all of the information relevant to the current state of the game is displayed on the LCD, and the games consist of rounds of 3 games each.

In this game there are several components involved, three buttons that can be pressed by the players, one buzzer that signals the start of the round, and an LCD display and potentiometer to output the current state of the game to the players.

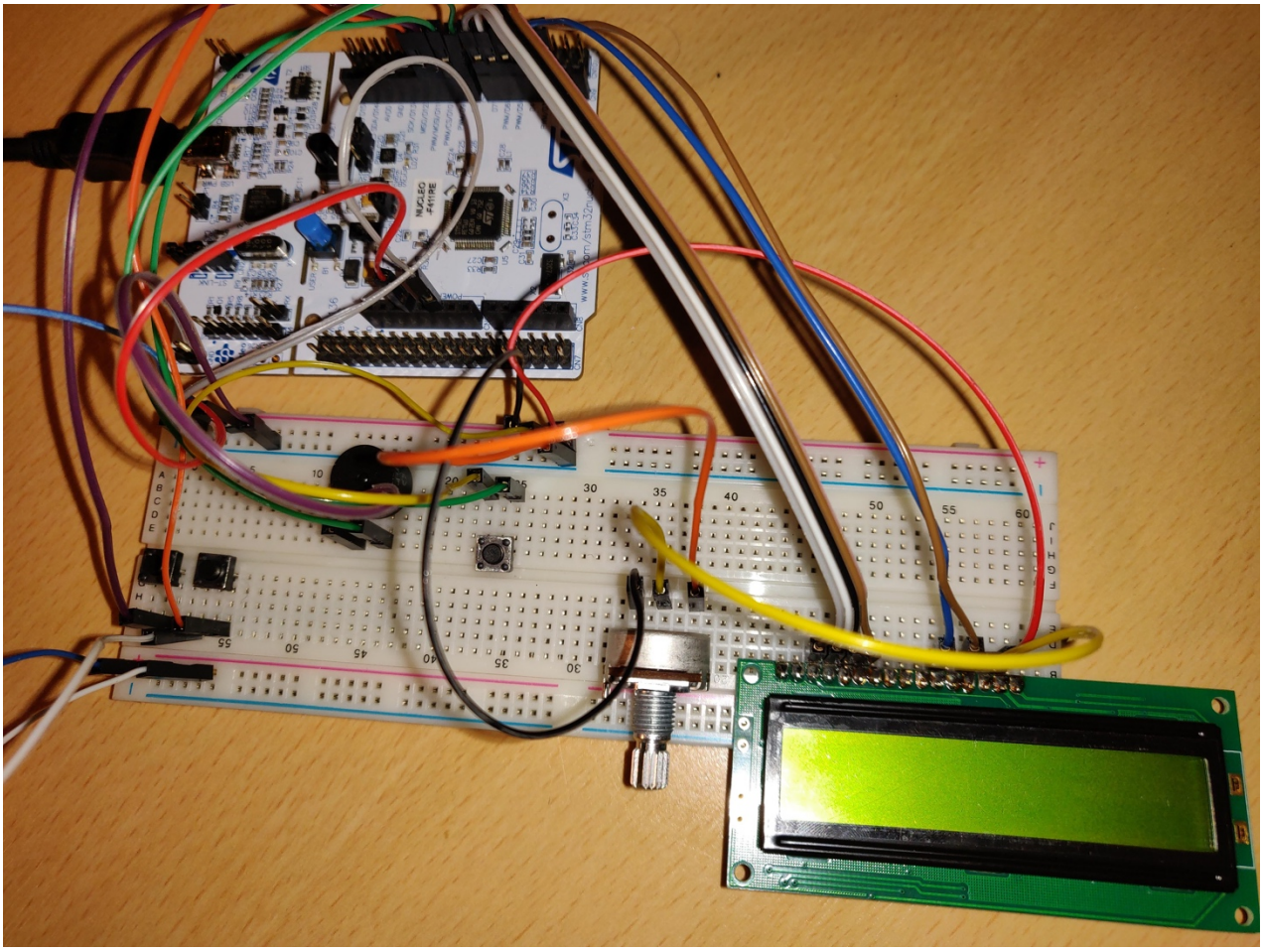
In order to keep track of the reaction times we will use timers, which are supported by this architecture as well as interrupts to handle each of the presses from the players.

2. DIAGRAM

2.1 CIRCUIT



2.2 FINAL IMPLEMENTATION



3. PROGRAM

Source code, also available in GitHub
https://github.com/UnZurdo/CS3514/blob/master/main.cpp
Most of the debugging messages that are outputted through the serial port have been deleted from the final version.

```
#include "mbed.h"
#include "buzzer.h"
#include <time.h>
#include "TextLCD.h"

#define ROUND_TIME 5 //minimum time between each round 5s
#define BUZZER_TIME 3 //minimum time for the buzzer 3s
#define UNDEFINED 14000 // value used to represent an undefined time
difference
#define PIN_BUZZER D8
#define PIN_START D9
#define PIN_P1 D10
```

```
#define PIN_P2 D11
#define PINS_LCD D2,D3,D4,D5,D6,D7

// Pins for LCD display D2, D3, D4, D5, D6, D7
TextLCD lcd(PINS_LCD); //rs, e, d4, d5, d6, d7

// Output interface to serial port connected to computer
Serial pc(USBTX, USBRX);

Beep buzzer(PIN_BUZZER);
DigitalIn startButton(PIN_START); //start button
InterruptIn p1Button(PIN_P1); //player1 button
InterruptIn p0Button(PIN_P2); //player2 button

// Global variables
double timeP1, timeP0, sumP1, sumP0, t1, t2;
int winsP1,winsP0;

// Global timer for the game
Timer t;

// Function to handle each interrupt
void playerInterrupt(){
    // Reads current value of the timer
    t2 = t.read_ms();
    double diff = t2 - t1;
    // If player 1 presses the button for the first time in this round
    if(p1Button && timeP1 == UNDEFINED) {
        if (t1 == UNDEFINED) {
            pc.printf("Player1 => DISQUALIFIED\n", diff);

        } else {
            pc.printf("Player1 => Time diff: %.0f ms\n", diff);
            timeP1 = diff;
        }
    }
    // If player 2 presses the button for the first time in this round
    else if (p0Button && timeP0 == UNDEFINED){
        if (t1 == UNDEFINED) {
            pc.printf("Player0 => DISQUALIFIED\n", diff);

        } else {
            pc.printf("Player0 => Time diff: %.0f ms\n", diff);
            timeP0 = diff;
        }
    }
}

void updateGame(){
    // In case of DISQUALIFIED; the player will be penalized,
    // adding the maximum reaction time possible
    sumP1 += timeP1;
    sumP0 += timeP0;

    if (timeP1 <= timeP0 ){
        winsP1++;
    } else {
```

```
        winsP0++;
    }
    timeP1 = UNDEFINED;
    timeP0 = UNDEFINED;

}

int main() {
      /* initialize random seed: */
    srand (time(NULL));

    int round = 1;
    // Set PullDown modes for each of the input buttons
    startButton.mode(PullDown);
    p1Button.mode(PullDown);
    p0Button.mode(PullDown);

    // Associate subrutine to interrupts triggered by players
    p1Button.rise(&playerInterrupt);
    p0Button.rise(&playerInterrupt);

    while(1) {
        // Game starts
        t.start();
        if(startButton == 1){
            // Reset variables and clean LCD display
            lcd.cls();
            round = 1;
            timeP1 = UNDEFINED;
            timeP0 = UNDEFINED;
            sumP1 = 0;
            sumP0 = 0;
            winsP1 = 0;
            winsP0 = 0;

            while(round <= 3){
                // Time for current round
                int delay = rand() % 10 + BUZZER_TIME;

                // Clear and print into display
                lcd.cls();
                lcd.printf("Round %d STARTS\n", round);


                /* Players shouldn't press the button before buzzer
                *  sounds ( while t1 == UNDEFINED )
                */ if they do so, they will be penalized
                t1 = UNDEFINED;
                wait(delay);

                // Buzzer sounds for 1 second at a given frequency
                buzzer.beep(261,1);

                // set actual timer for this round
                t.reset();
                t1 = t.read_ms();
```

```
                    // Clear and print into display
                    lcd.cls();
                    // if both players are disqualified, no one wins
                    if (timeP1 == UNDEFINED && timeP0 == UNDEFINED){
                        lcd.printf("both players \n DISQUALIFIED", round);

                    }
                    // One of the players isn't disqualified,
                   // so there is a winner
                    else {
                        lcd.printf("P0:%.0f  P1:%.0f\nP%d WINS by %.0f",
                           timeP0,timeP1, timeP1 <= timeP0, abs(timeP1 - timeP0));

                   }
                    // wait until ROUND finishes
                    wait(ROUND_TIME);

                    // update state for the next ROUND
                    ++round;
                    updateGame();

                }
                // Show winner, reaction times and reset state for new GAME
                lcd.printf("//END// P%d WINS\nP0: %.0f, P1: %.0f",
                    winsP1 < winsP0, sumP0/3.0, sumP1/3.0);

            }
        }
}
```

3.1 COMMENTS

The program makes use of three inputs, one button for each player and the start button, all these are instantiated as PullDown buttons, allowing us to interpret correctly each press without any external influence and without adding any additional hardware. As for the outputs, it uses one buzzer as well as additional classes to communicate with the serial port and the LCD display.

In order to generate random numbers, it uses the srand() and rand() functions, the initial seed will be unique each time the program runs ,as we are using the internal timer that comes included with the microcontroller, we could have used as well a value received through the ADC.

In this cases we don't have to read from all of the ports at the same time, as we are handling each individual press with an interruption, but if we had implemented it, we should have instantiated a new PortIn class, using PrtA and mapping the corresponding pins D9,D10 with a mask and reading both values at the same time using:

```
int pins = p.read()  // or the overloaded = operator
```

and then applying a mask again to identify which pin is up, doing a bitwise *and* operation.

Each time we write into the LCD display it is necessary to either rewrite the new line into the previous one or clearing the display using `lcd.cls()`, it is also important to consider the maximum length of the lines, so that the messages are formatted accordingly.

To keep track of the time difference between each press, it is necessary to implement a timer, that once it is starts at the beginning of the program, it will be used to read the current time in milliseconds and compute the difference.

3.2 EXECUTION

The program runs the game indefinitely, each time the players press the start button a new game is launched, each player can press the button only one time each round, if they press it before the buzzer sounds, they are disqualified, the penalization for being disqualified consists on being automatically added the maximum amount of reaction time possible to your counter and losing the round.
After a game is started, it won't be possible to stop it or re-start it unless we reset the microcontroller, losing all of the information about the current game.

4. CONCLUSION

This implementation makes use of interrupts to handle most of the user's input, this supposes a small additional overhead on the number of instructions executed and context switch performed in order to handle the interrupts, but as the total number of interruptions and time spent handling them is insignificant compared to the rest of the program, it suppose a great improvement compared to the process of polling a variable all the time.

4.1 DRAWBACKS

- The microcontroller makes an active wait each time you want to start a new Game until the start button is pressed, it would be possible to implement another interrupt subroutine that changes the state of the system, setting a global variable *game := True*, meanwhile the main process would be waiting for a conditional variable to be met, in this case, for the game to start. But as the microcontroller may not support this kind of synchronization construct it was decided to keep the current implementation.
- Due to the nature of interrupts and the implementation supported by this architecture, there is a case that it is not supported by this program, that being when both players press the button at the same time, in that case the second interrupt could be discarded and it wouldn't be possible for a tie to ever occur, to fix this we should read both ports at the same time, instead of one of the buttons, but this is very unlikely to happen so it wasn't supported by this implementation.

- It could be possible to optimize more the amount of memory usage by using shorter variables for some of the counters like the number of rounds and wins.

4.2 IMPROVEMENTS

- Handles the case of DISQUALIFIED, if one of both of the players press their respective buttons before the buzzer sounds one of the edge cases will handle this new outcome, penalizing the player/s and informing them through the LCD display.
- If one of the players press the button and the round finishes, they will be penalized as well.
- The program makes use of interruptions to handle the user input, instead of making an active wait, the microcontroller will execute a subroutine ("playerInterrupt"), which will update the current state of the program.
- Once one of the players presses the button, they aren't allowed to press it again, ignoring any future press that follows the first one.
- All of the constants are declared in the beginning as directives.