DESIGN OF WAL a Mol USING PIC16F877A

Sudeep Saurabh

Swapnil Gour

CONTENT

- References

INTRODUCTION

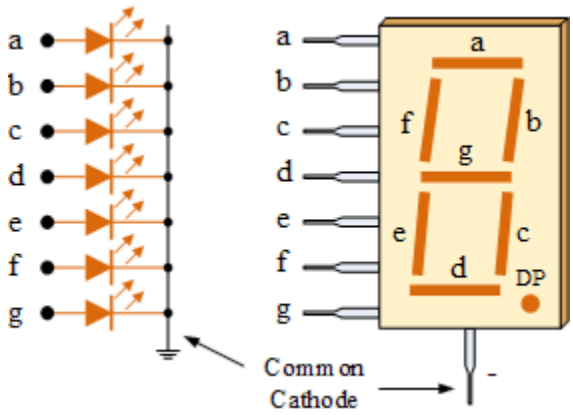
This project is a simple PIC game where LEDs go on randomly one at a time. The objective is to “trap” the LED among 4 LEDs which are turning ON randomly. We need to press the button corresponding to that particular LED which is glowing at that instant of time. For every catch, the score, displayed on a seven-segment display, is incremented. As the game proceeds the speed of the LEDs increases every time when the score goes beyond multiples of 5 (5,10, 15, 20). And there is a reset button which resets the score back to zero and speed of LEDs reduces to initial (starting point) condition.

We simulated this game on PROTEUS 8 software using four LEDs, four buttons and a dual 7 segment display. PIC16F877A microcontroller is used. This project is coded using XC8.

COMPONENTS

Here is the list of components we have used for the simulation of our project. And brief about each component:

1. Dual Seven-segment Display - Common Cathode



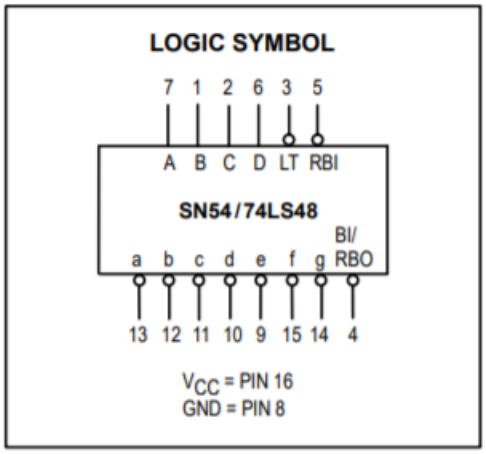
The 7-segment display, also written as “seven segment display”, consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a numerical digit (both Decimal and Hex) to be displayed.

The displays common pin is generally used to identify which type of 7-segment display it is. As each LED has two connecting pins, one called the “Anode” and the other called the “Cathode”.

The Common Cathode (CC) – In the common cathode display, all the cathode connections of the LED segments are joined together to logic “0” or ground. The individual segments are illuminated by application of a “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the individual Anode terminals (a-g).

Common Cathode 7-segment Display

2. 74LS48

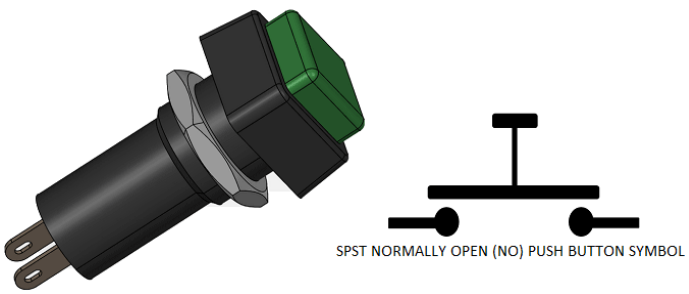


The 74LS48 (also SN54) is a BCD to 7-Segment Decoder that is used to display numbers decoded in binary coded decimal format. The 7-Segment is a small seven LED-based device use to represent a single numeric value from 0 to 9. Each 7-Segment has seven input pins to light up a single led in the seven segments. Every time to make a single number some specific pins should have power input.

It consists of NAND gates, input buffers and seven AND-OR-INVERT gates. Seven NAND gates and one driver are connected in pairs to make BCD data and its complement available to the seven decoding AND-OR-INVERT gates. The remaining NAND gate and three input buffers provide lamp test, blanking input/ripple-blanking input for the LS48.

- The circuit accepts 4-bit binary-coded-decimal (BCD) and, depending on the state of the auxiliary inputs, decodes this data to drive other components.
- The IC is a TTL based device, which allows it to be controlled by any TTL device or microcontroller. It is used to control common cathode 7-Segments.

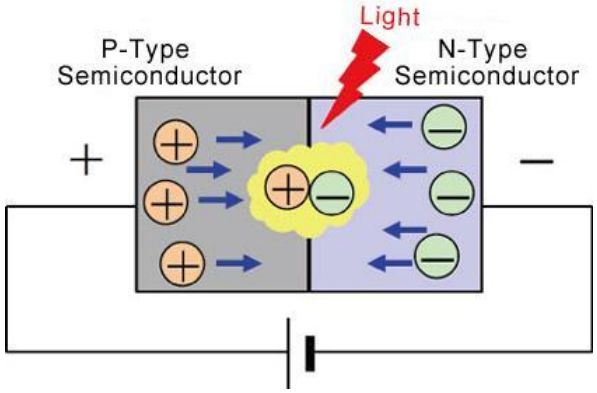
3. SPST Pushbuttons (x 5)



SPST = Single Pole, Single Throw

A Single Pole Single Throw (SPST) switch is a switch that only has a single input and can connect only to one output. This means it only has one input terminal and only one output terminal. A Single Pole Single Throw switch serves in circuits as on-off switches. This type can be used to switch the power supply to a circuit.

4. LEDs (x 4)



In the simplest terms, a light-emitting diode (LED) is a semiconductor device that emits light when an electric current is passed through it. Light is produced when the particles that carry the current (known as electrons and holes) combine together within the semiconductor material. Since light is generated within the solid semiconductor material, LEDs are described as solid-state devices.

5. 10k ohm and 200 ohm 1/4 W Resistors (x 5 each)

These are your run-of-the-mill 1/4 Watt, +/- 5% tolerance PTH resistors. Commonly used in breadboards and other prototyping applications, these 10K ohm resistors make excellent pull-ups, pull-downs and current limiters. These thick-lead versions of the resistors fit snugly into a breadboard with very little movement.

6. 4 MHz Crystal

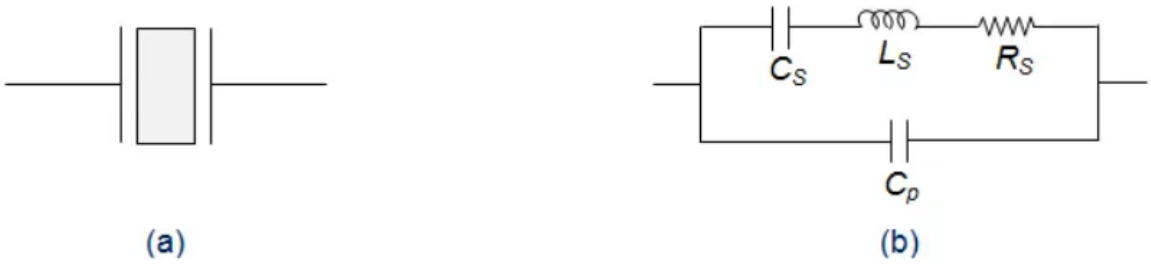


Figure 1 (a) Quartz Crystal (b) Equivalent Electric Circuit

Crystal oscillators operate on the principle of inverse piezoelectric effect in which an alternating voltage applied across the crystal surfaces causes it to vibrate at its natural frequency. It is these vibrations which eventually get converted into oscillations.

In crystal oscillators, the crystal is suitably cut and mounted between two metallic plates as shown by Figure 1a whose electrical equivalent is shown by Figure 1b. In reality, the crystal behaves like a series RLC circuit.

In general, the frequency of the crystal oscillators will be fixed to be the crystal's fundamental or characteristic frequency which will be decided by the physical size and shape of the crystal.

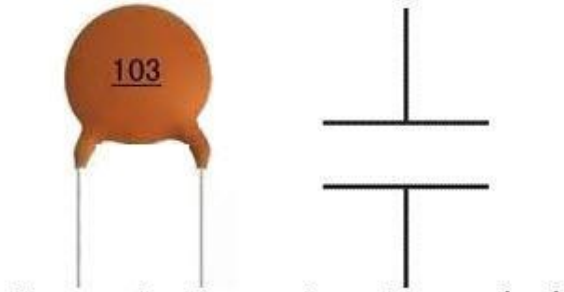
Advantages:

- They have a high order of frequency stability.
- The quality factor (Q) of the crystal is very high.

Disadvantages:

- They are fragile and can be used in low power circuits.
- The frequency of oscillations cannot be changed appreciably.

7. 22pF Ceramic Capacitor (x 2)



Ceramic Capacitor & symbol

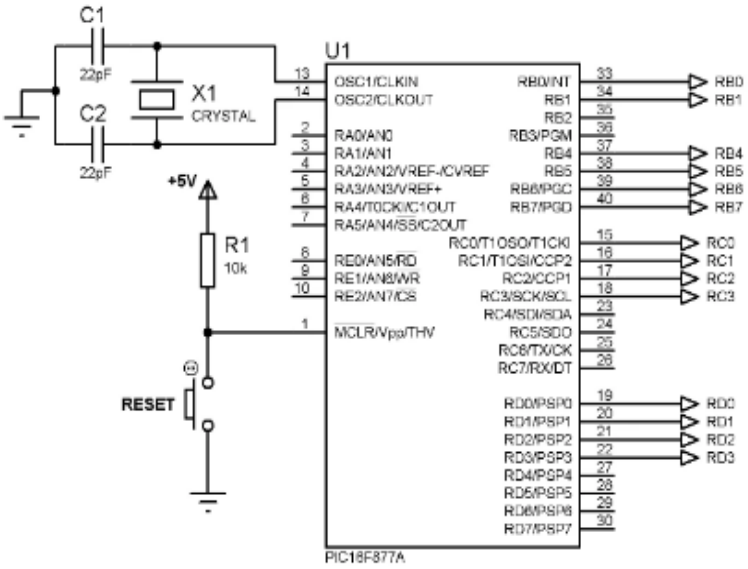
A ceramic capacitor is a fixed-value capacitor where the ceramic material acts as the dielectric. It is constructed of two or more alternating layers of ceramic and a metal layer acting as the electrodes. The composition of the ceramic material defines the electrical behavior and therefore applications.

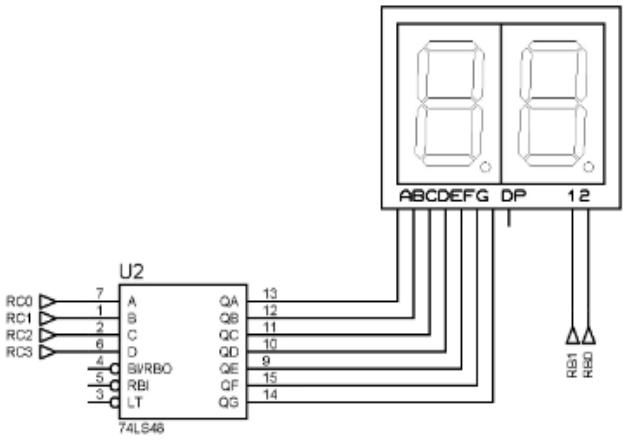
Ceramic capacitors are most commonly found in every electrical device and it uses a ceramic material as the dielectric. The ceramic capacitor is a non-polarity device, which means they do no have polarities. So we can connect it in any direction on a circuit board. For this reason, they are generally much safer than electrolytic capacitors. Many types of capacitors, such as the tantalum bead do not have a polarity.

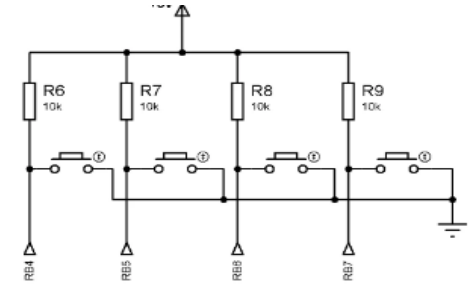
Advantages of ceramic capacitors:

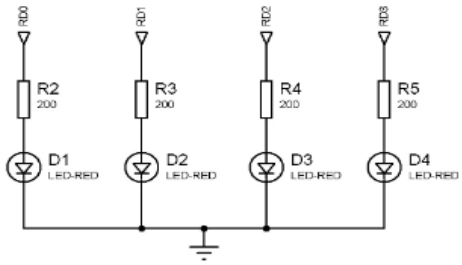
- Manufacturing cost is less
- High-frequency performance is exhibited
- The stability of the capacitor is dependent on the ceramic dielectric

SCHEMATIC

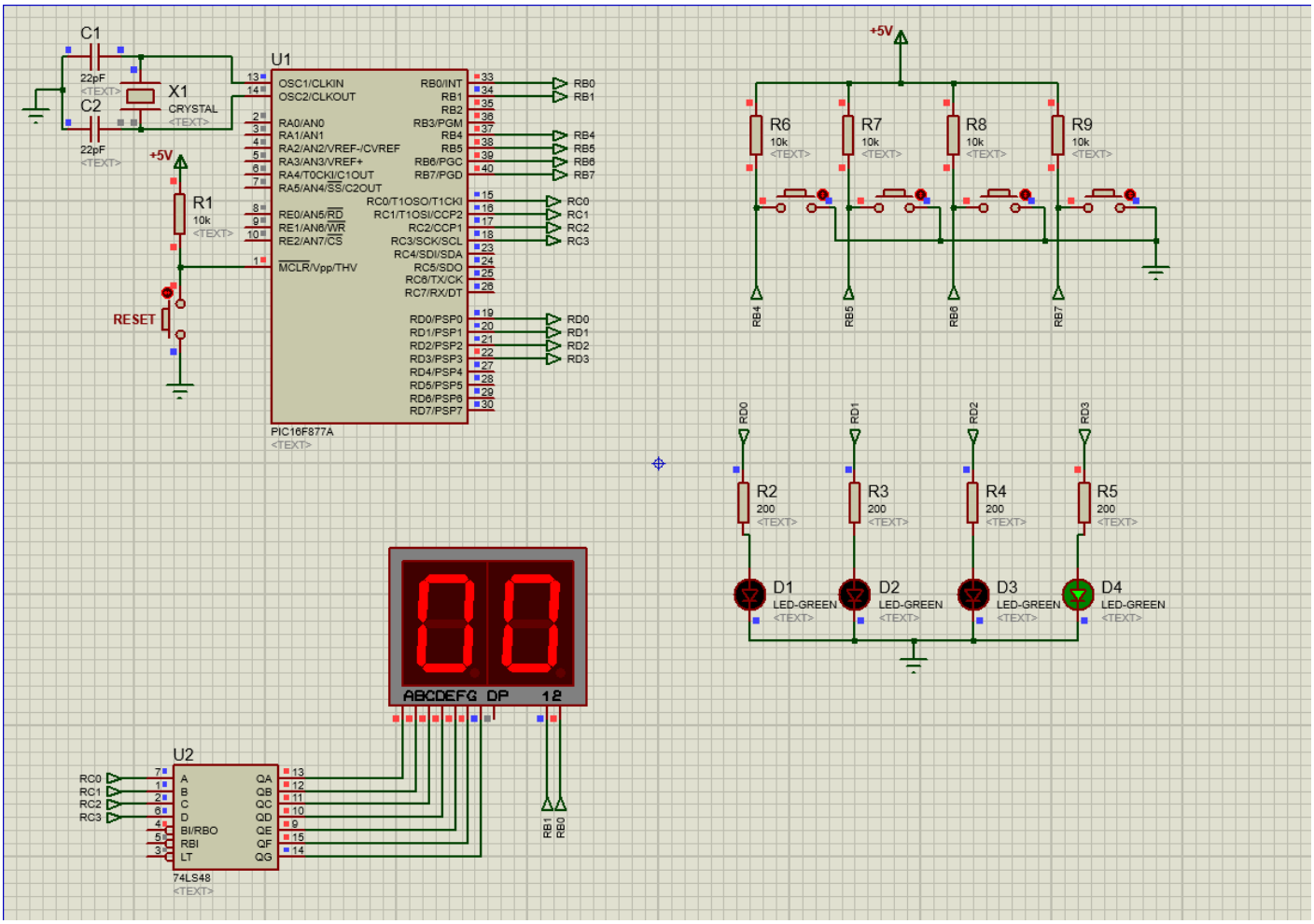

C1
22pF
C2
22pF
X1
CRYSTAL
U1
OSC1/CLKIN
OSC2/CLKOUT
RA0/AN0
RA1/AN1
RA2/AN2/VREF-/CVREF
RA3/AN3/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/C2OUT
RE0/AN5/RD
RE1/AN6/WR
RE2/AN7/CS
MCLR/VPP/THV
RB0/INT
RB1
RB2
RB3/PGM
RB4
RB5
RB6/PGC
RB7/PGD
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0
RD1/PSP1
RD2/PSP2
RD3/PSP3
RD4/PSP4
RD5/PSP5
RD6/PSP6
RD7/PSP7
PIC18F877A
+5V
R1
10k
RESET
RB0
RB1
RB4
RB5
RB6
RB7
RC0
RC1
RC2
RC3
RD0
RD1
RD2
RD3


ABCDEFG DP
12
U2
A
B
C
D
BI/RBO
RBI
LT
QA
QB
QC
QD
QE
QF
QG
74LS48
RC0
RC1
RC2
RC3
RB1
RB0


R6
10k
R7
10k
R8
10k
R9
10k
RB4
RB5
RB6
RB7


RD0
RD1
RD2
RD3
R2
200
R3
200
R4
200
R5
200
D1
LED-RED
D2
LED-RED
D3
LED-RED
D4
LED-RED

COMPONENTS SETUP ON PROTEUS



SIMULATION RESULT

Complete Simulation

This game works similarly like Whac a mole. In this game we have to switch the particular key below which the light blinks. Blinking of light becomes faster as one progresses in the level.

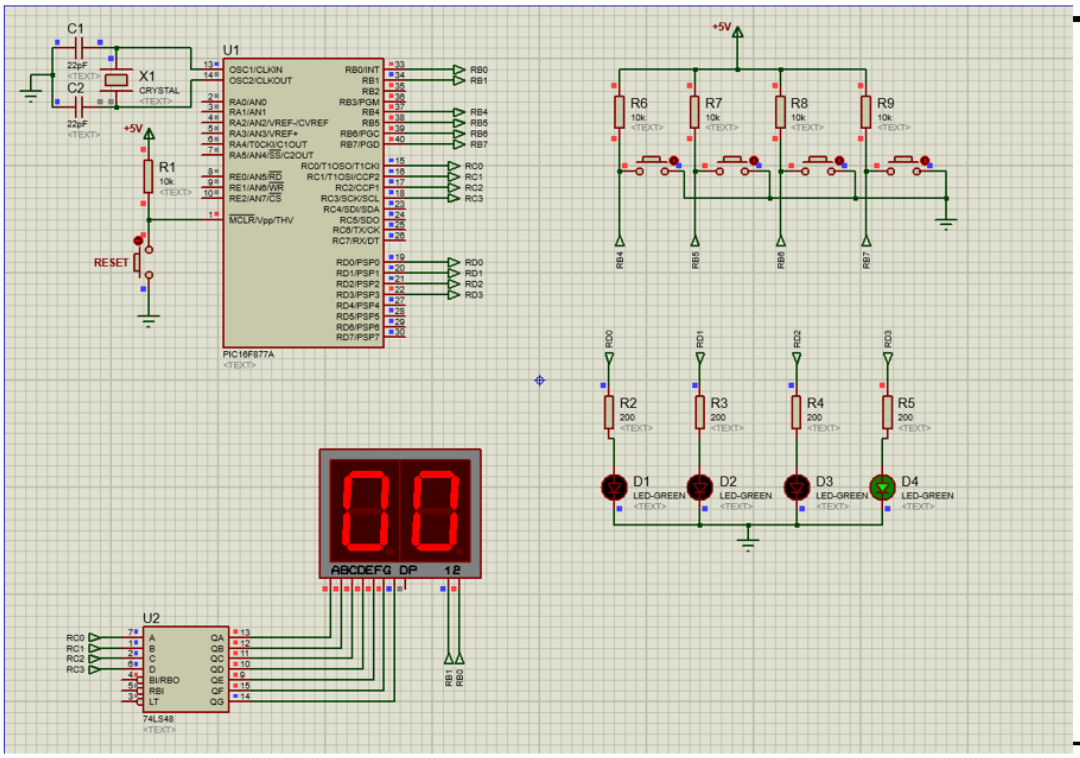
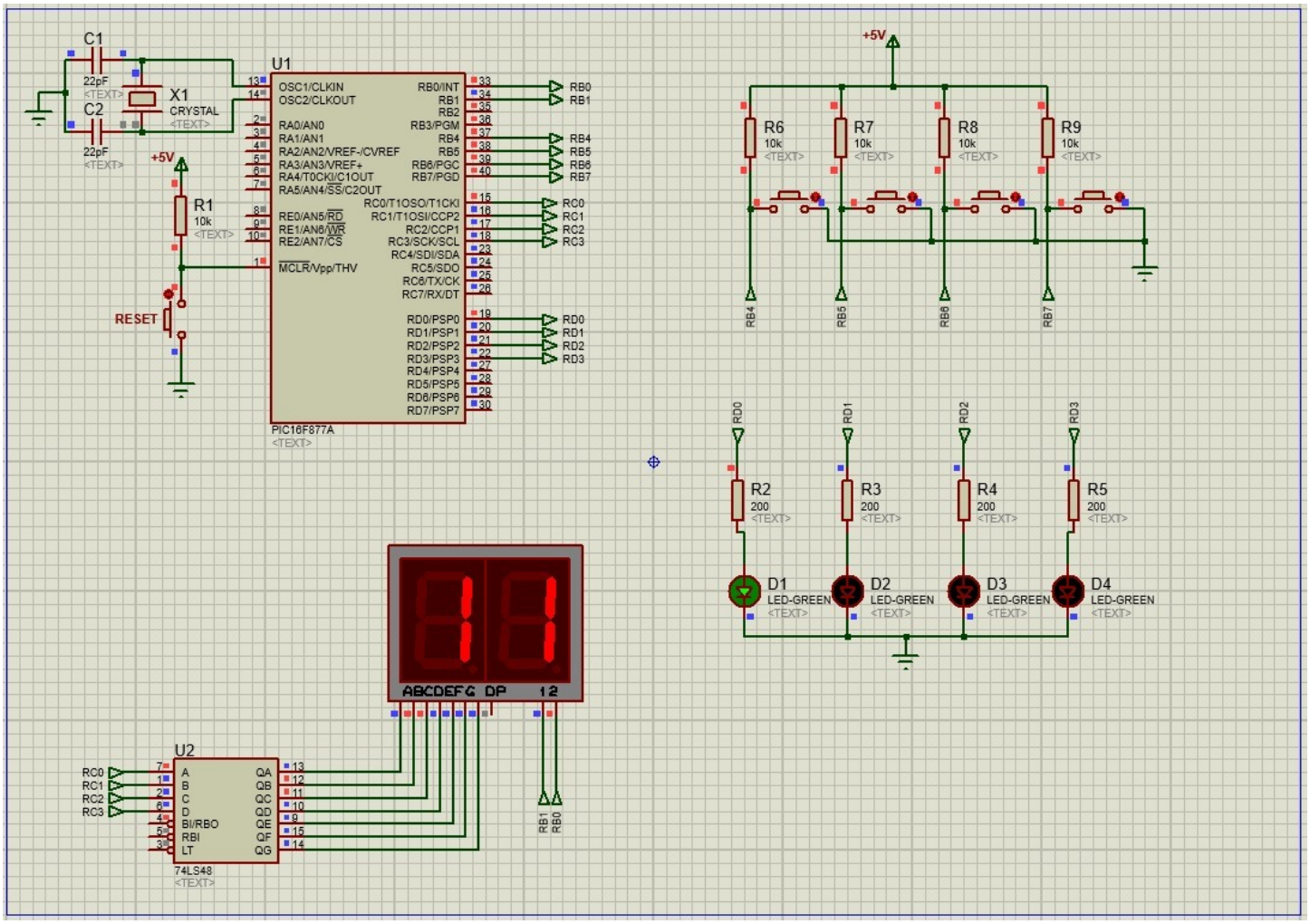


Figure on LHS shows the game has started with initial score zero



Snapshot from simulation when score have reached 11

CODE

```
/*
 * File:  led_game.c
 * Blinking LED Game
*/


#pragma config FOSC = XT        // Oscillator Selection bits (XT oscillator)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = OFF      // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF        // Low-Voltage (Single-Supply) In-Circuit Serial Programming
Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit (Data EEPROM code
protection off)
#pragma config WRT = OFF        // Flash Program Memory Write Enable bits (Write protection
off; all program memory may be written to by EECON control)
#pragma config CP = OFF         // Flash Program Memory Code Protection bit (Code protection
off)


#define _XTAL_FREQ 4000000
#include <xc.h>

#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int rand_num;
int old_rand_num;
int score;
int score_tens;
int score_ones;
int shifter;
int delay;


void someDelay(){
    switch(delay){
        case 500:
            __delay_ms(500);
            break;
        case 200:
            __delay_ms(200);
            break;
        case 100:
            __delay_ms(100);
            break;
        case 50:
            __delay_ms(50);
            break;
    }
    return;


}
void main(void) {

    T1CON = 0b00000001;             //turn on TMR1
    INTCON = 0b10101000;            //use global, TMR0 and RB change interrupts
    OPTION_REG = 0b00000100;        //use 1:32 prescale for TMR0
```

```
    TRISB = 0xF0;                   //inputs on RB4 to RB7, outputs for RB0 to RB3
    TRISC = 0;                      //all PORTC are outputs
    TRISD = 0;                      //all PORTD are outputs

    delay = 500;                    //default delay

    loop:
    while(1){
        srand(TMR1);                //use TMR1 value as seed for random
        rand_num = rand() % 9;      //generate random numbers from 0 to 9

//filter out consecutive numbers, 0 and those that has at least two bits high (011 = 3, 101
= 5, 110 = 6, 111 = 7)

        if(rand_num == old_rand_num || rand_num == 0 || rand_num == 3 || rand_num == 5 ||
rand_num == 6 || rand_num == 7){
            goto loop;
        }
        PORTD = rand_num;
        old_rand_num = rand_num;    //
        someDelay();

  //a function that varies the delay. Because passing variables to __delay_ms() doesn't work
    }

    return;
}

void interrupt isr(void){
    if(TMR0IF){
         //switch digits for every interrupt. Read about POV for better understanding
            shifter++;
            switch(shifter){
                case 1:
                    PORTB = 0b11111110;
                    PORTC = score_ones;
                    break;
                case 2:
                    PORTB = 0b11111101;
                    PORTC = score_tens;
                    break;
                case 6:
                    shifter = 0;
                    break;
            }
            TMR0IF = 0;     //manual clearing of interrupt flag

    }
    if(RBIF){
        int dummy = PORTB;          //read PORTB to clear mismatch condition
        int masked = (PORTB & 0xF0) >> 4;   //acquire pressed buttons
        if(~masked == (PORTD + 0xFFF0)){
            score++;
            score_tens = score/10;
            score_ones = score - score_tens*10;
            //adjust speed according to score
            if(score <= 5){
                delay = 500;
            }else if(score > 5 && score <= 10){
                delay = 200;
            }else if(score > 10 && score <= 20){
                delay = 100;
            }else if(score > 20 && score <= 30){
```

```
                delay = 50;
            }else if(score > 99){
                score = 99;
            }else{
                delay = 500;
            }
        }
        RBIF = 0;
    }

}
```

CONCLUSION

We have successfully simulated this game on PROTEUS 8 software using four LEDs, four buttons and a dual 7 segment display. PIC16F877A microcontroller is used. This project is coded using MPLAB (XC8 compiler).

Further we can modify this game like combo where u get extra score one u start getting correct continues, and we can even do decrease the score if u get it wrong

THANK YOU

REFERENCES

[1] https://www.youtube.com
[2] https://www.techmemicro.com
[3] https://www.videoamusement.com
[4] https://www.instructables.com